Defence Research and Development Canada
Recherche et développement pour la défense Canada

# A Multi-Packet Signature Approach to Passive Operating System Detection

Annie De Montigny-Leboeuf

The work described in this document was sponsored by the Department of National Defence under Work Unit 15bf.

**Defence R&D Canada – Ottawa**
TECHNICAL MEMORANDUM
DRDC Ottawa TM 2005-018

**Communications Research Centre Canada**
TECHNICAL NOTE
CRC-TN-2005-001
January 2005

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

1. REPORT DATE
**JAN 2005**

2. REPORT TYPE

3. DATES COVERED
**-**

4. TITLE AND SUBTITLE
**A Multi-Packet Signature Approach to Passive Operating System Detection (U)**

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
**Defence R&D Canada -Ottawa,3701 Carling Ave,Ottawa Ontario,CA,K1A 0Z4**

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**The original document contains color images.**

14. ABSTRACT

**Remote operating system discovery can provide valuable contextual information regarding the computers connected to the network. In particular, operating system discovery can help identify potential vulnerable computers or may help prioritize alarms and responses in times of attack. The Network Security Research Group at the Communication Research Centre (CRC) has developed novel techniques for passive operating system discovery. The methodology developed allows derivation of a signature from a set of packets. The tests are conducted passively on regular traffic. They are non-intrusive and do not rely on access to application or user data. Because they are passive, the techniques do not consume bandwidth and do not disrupt network assets. Over a dozen tests have been developed to analyse headers of packets seen on a network. The tests are conducted on headers of various types of protocols: ARP, IP, ICMP, UDP and TCP. This document describes the tests in detail. They have been implemented in a prototype written in JAVA, which includes a database containing the "fingerprints" of almost 200 versions of operating systems. The prototype was used to collect these signatures from our testbed and was also used on real user traffic for preliminary evaluation of the tests’ performance.**

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | | **182** | |

# A Multi-Packet Signature Approach to Passive Operating System Detection

Annie De Montigny-Leboeuf
Communications Research Centre Canada

The work described in this document was sponsored by the Department of National Defence under the Work Unit 15bf

**Defence R&D Canada – Ottawa**

Technical Memorandum

DRDC Ottawa TM 2005-018

**Communications Research Centre Canada**

Technical Note

CRC-TN-2005-001

January 2005

# Abstract

Remote operating system discovery can provide valuable contextual information regarding the computers connected to the network. In particular, operating system discovery can help identify potential vulnerable computers or may help prioritize alarms and responses in times of attack. The Network Security Research Group at the Communication Research Centre (CRC) has developed novel techniques for passive operating system discovery. The methodology developed allows derivation of a signature from a set of packets. The tests are conducted passively on regular traffic. They are non-intrusive and do not rely on access to application or user data. Because they are passive, the techniques do not consume bandwidth and do not disrupt network assets. Over a dozen tests have been developed to analyse headers of packets seen on a network. The tests are conducted on headers of various types of protocols: ARP, IP, ICMP, UDP and TCP. This document describes the tests in detail. They have been implemented in a prototype written in JAVA, which includes a database containing the "fingerprints" of almost 200 versions of operating systems. The prototype was used to collect these signatures from our testbed and was also used on real user traffic for preliminary evaluation of the tests' performance.

# Résumé

La capacité de reconnaître à distance les systèmes d'exploitation peut permettre d'acquérir de l'information contextuelle et précieuse à propos des ordinateurs connectés à un réseau. En particulier la reconnaissance des systèmes d'exploitation peut permettre d'identifier des ordinateurs potentiellement vulnérables ou peut contribuer à prioriser les alarmes et les réactions en cas d'attaques. Le groupe de recherche en sécurité des réseaux au Centre de recherche sur les communications (CRC) a développé de nouvelles techniques pour la reconnaissance passive des systèmes d'exploitation. La méthodologie développée permet d'établir des signatures s'étalant sur plusieurs paquets. Les tests reposent sur des techniques passives et non-intrusives d'analyse de trafic régulier. L'accès aux données provenant des applications et des usagers n'est pas requis. Les techniques étant passives, elles ne consomment pas de bande passante et ne perturbent pas les composantes du réseau. Plus d'une douzaine de tests ont été conçus pour analyser les entêtes des paquets circulant sur le réseau. Les protocoles dont les entêtes sont examinées sont: ARP, IP, ICMP, UDP et TCP. Ce document décrit en détail les différents tests développés. Ces tests ont été implémentés dans un prototype, écrit en JAVA, qui contient une base de données comprenant les «empreintes» de près de 200 versions de systèmes d'exploitation. Le prototype a permis de recueillir ces signatures sur un réseau de test ainsi qu'à évaluer les différents tests avec du trafic d'usagers véritables.

This page intentionally left blank.

# Executive summary

A tool that has the ability to identify the operating system (OS) version of computers connected to a network is useful to both network managers and security analysts charged with protecting the network. An operating system identification tool can provide significant contextual information, and is even more valuable if the tool itself doesn't disrupt network traffic and can't normally be detected.

The Network Security Research Group at the Communication Research Centre (CRC) has developed a series of tests for passively detecting operating systems, and has implemented a prototype software tool as a proof of concept. The tool is completely passive as it does not generate any probe or trigger packets. The approach taken is based on the analysis of packet headers at the data-link, network, and transport layers, thus the tool does not rely on access to application data. The methodology goes beyond individual packet analysis commonly used in open source and commercial operating system identification tools. Because certain packets have influence on subsequent packets, some information can only be gained when related packets are analysed together. The uniqueness of this approach is in the use of lightweight state-aware mechanisms to derive signatures from multiple packets. Over a dozen tests have been developed to analyse headers of packets seen on a network. The tests are conducted on headers of various types of protocols: ARP, IP, ICMP, UDP, and TCP. A number of these tests are adaptations of active techniques, i.e. techniques that normally require a form of interaction.

The passive OS detection tool was programmed in Java and is one of a set of tools developed by the team for network monitoring and analysis. It includes a database containing the signatures of close to 200 versions of operating systems among the most popular OS families (Linux, SunOS, MacOS, Windows, FreeBSD, OpenBSD, NetBSD, Novell, BeOS, and QNX). When a test produces a signature that cannot be found in the database, a mechanism to look for an alternative signature is called upon. A module manages the information coming from all the tests and attempts to identify the set of possible operating systems on which the individual tests agree.

The prototype includes automated learning capabilities, verification capabilities, and a regular mode of operation during which the tool performs passive OS detection on live traffic or pre-recorded traffic traces.

The signatures contained in the database were obtained using the tool in a controlled environment. Target operating systems were installed and queried methodically in the local testbed and the prototype was used to collect and store the signatures observed. This helped achieving control and uniformity during the capture process.

This document describes the OS identification techniques implemented in the tool, the signature collection process, and some preliminary results obtained on real user traffic. The key elements of these signatures are described in detail in the core document and the signatures for all tests are provided in Annex. Some operating systems have very distinct signatures that allow the prototype to recognize them easily, but in many cases it is the combination of the tests that leads to a small subset of possible guess.  Some peculiar behaviour has been observed during this study, not all of which could be thoroughly tested.  The testing process and prototype itself would benefit from a number of modifications and extensions.  While little resource at this time is being put on enhancing the program, the signature database is being updated as new operating systems are released.  The prototype is in a stage where it is considered a proof of concepts.  The tool works as a standalone application and was also integrated into in-house information gathering tools.  The concepts are being considered to complement a Scenario-Driven Intrusion Detection System under development within the team.

A. De Montigny-Leboeuf.  2004.  A Multi-Packet Signature Approach to Passive Operating System Detection. DRDC OTTAWA TM 2005-018 / CRC-TN-2005-001. Communications Research Centre Canada. Defence R&D Canada - Ottawa

# Sommaire

Un outil qui a la capacité d'identifier la version du système d'exploitation (SE) des ordinateurs reliés à un réseau est utile aux administrateurs de réseau ainsi qu'aux analystes de sécurité responsables de la protection du réseau.  Un tel outil peut fournir de l'information contextuelle, et est davantage utile s'il ne perturbe pas le trafic de réseau et ne peut être détecté.  Le groupe de recherche en sécurité des réseaux au centre de recherche sur les communications (CRC) du Canada a développé une série de tests pour détecter passivement les systèmes d'exploitation.  Le groupe a développé un prototype pour valider le concept.  L'outil est complètement passif car il ne produit aucun trafic pour sonder le réseau.  L'approche adoptée est basée sur l'analyse des en-têtes de paquets aux niveaux des couches de liaison de données, de réseau, et de transport.  L'outil ne nécessite donc pas l'accès aux données d'applications.  La méthode dépasse l'analyse individuelle des paquets, méthode d'identification passive des systèmes d'exploitation généralement utilisée dans les logiciels libres (« open source ») et commerciaux.  Or, puisque certains paquets transmis ont de l'influence sur les paquets ultérieurs, certaines informations ne peuvent être obtenues qu'en analysant les paquets ensemble.  La particularité de l'approche adoptée est dans l'utilisation de mécanismes simples mais capable de dériver des signatures s'étendant sur plusieurs paquets.  Un peu plus d'une douzaine de tests ont été développés pour analyser les en-têtes des paquets circulant sur un réseau.  Les tests sont effectués sur divers types d'en-têtes de protocole : ARP, IP, ICMP, UDP, et TCP.  Un nombre de ces tests sont des adaptations de techniques actives, c'est-à-dire qui requièrent normalement une forme d'interaction.

L'outil passif de détection de SE a été programmé en Java et fait partie d'un ensemble d'outils développés par l'équipe pour la surveillance et l'analyse de réseau.  Il inclut une base de données contenant les signatures de près de 200 versions de système d'exploitation parmi les familles les plus populaires (Linux, SunOS, MacOS, Windows, FreeBSD, OpenBSD, NetBSD, Novell, BeOS, et QNX).  Quand un test produit une signature qui ne peut pas être trouvée dans la base de données, un mécanisme pour rechercher une signature alternative est appelé.  Un module contrôle l'information venant de tous les tests et essaye d'identifier l'ensemble des systèmes d'exploitation possibles sur lequel les différents tests sont en accord.

Le prototype inclut une capacité d'apprentissage automatisé, de vérification, ainsi qu'un mode de fonctionnement régulier permettant la détection passive de SE en direct ou sur des traces de trafic pré-enregistrées.  Les signatures contenues dans la base de données ont été obtenues à l'aide de l'outil dans un environnement contrôlé.  Les systèmes d'exploitation ont été installés et testés méthodiquement dans le banc d'essai et le prototype a été employé pour recueillir et stocker les signatures.  Ceci a aidé à maintenir un niveau de control et d'uniformité au cours du processus de collection des signatures.

Ce document décrit les techniques d’identification de systèmes d’exploitations développées au cours de ce travail, le processus de collection des signatures, et quelques résultats préliminaires obtenus à partir de trafic d’usagers réels. Les éléments clés de ces signatures sont décrits en détail dans le coeur du document et les signatures pour tous les tests sont fournies en annexe. Quelques systèmes ont des signatures très distinctes qui permettent au prototype d’identifier facilement leurs systèmes d’exploitation, mais dans beaucoup de cas, c'est la combinaison des tests qui mène à un petit sous-ensemble de possibilités. Nous avons observé durant cette étude quelques comportements particuliers, certains d’entre eux n’ont pu être examinés de façon exhaustive. Le processus d’évaluation et le prototype lui-même tireraient bénéfice d'un certain nombre de modifications et d’enrichissements. Tandis que peu de ressources sont actuellement affectées à l’amélioration du programme, la base de données des signatures est toutefois mise à jour lorsque de nouveaux systèmes d’exploitation sont mis en circulation. Le prototype est dans un état validant le principe. L'outil fonctionne comme application autonome et a également été intégré dans des outils développés à l’interne pour la surveillance et l'analyse de réseau. L’équipe est présentement à considérer l’applicabilité des principes utilisés pour détecter les systèmes d’exploitation pour complémenter un système de détection d'intrusion fondé sur des scénarios.

A. De Montigny-Leboeuf. 2004. A Multi-Packet Signature Approach to Passive Operating System Detection. DRDC OTTAWA TM 2005-018 / CRC-TN-2005-001. Centre de recherche sur les communications Canada. R & D pour la défense Canada - Ottawa

# Table of contents

Abstract ..... i

Executive summary ..... iii

Sommaire ..... v

Table of contents ..... vii

List of figures ..... x

Acknowledgements ..... xii

1. Introduction ..... 1

2. Background ..... 3

2.1 Active Fingerprinting Techniques ..... 3

2.2 Passive Fingerprinting Techniques ..... 5

2.3 Header fields used in fingerprinting ..... 6

2.3.1 IP Don’t Fragment bit ..... 6

2.3.2 IP Time-To-Live ..... 6

2.3.3 IP Service Type ..... 7

2.3.4 IP Identification ..... 7

2.3.5 ARP Target Hardware address ..... 8

2.3.6 ICMP code ..... 8

2.3.7 ICMP Identifier and ICMP Sequence Number ..... 9

2.3.8 Data of ICMP Echo messages ..... 10

2.3.9 Data of ICMP Error messages ..... 10

2.3.10 TCP Sequence Number ..... 11

2.3.11 TCP Acknowledgment Number ..... 11

2.3.12 TCP Flags ..... 11

2.3.13 TCP Reserved ..... 13

2.3.14 TCP Window ..... 14

2.3.15 TCP Options ..... 14

3. Passives Tests Developed ..... 19
3.1 Categories of Passive Tests ..... 19
3.1.1 Singleton ..... 19
3.1.2 Sample ..... 20
3.1.3 Stimulus-Response ..... 20
3.2 Tests Descriptions ..... 21
3.2.1 PassiveTest_TCP_SYN (Singleton) ..... 21
3.2.2 PassiveTest_ARP_Request (Singleton) ..... 22
3.2.3 PassiveTest_TCP_ISN (Sample) ..... 23
3.2.4 PassiveTest_IP_ID (Sample) ..... 26
3.2.5 PassiveTest_TCP_TS (Sample) ..... 30
3.2.6 PassiveTest_ARP_Retransmit (Sample) ..... 32
3.2.7 PassiveTest_ICMP_ID_SEQ (Sample) ..... 33
3.2.8 PassiveTest_TCP_SYNACK (Stimulus-Response) ..... 37
3.2.9 PassiveTest_TCP_RSTACK (Stimulus-Response) ..... 39
3.2.10 PassiveTest_ICMP_Unreach (Stimulus-Response) ..... 40
3.2.11 PassiveTest_ICMP_Echo (Stimulus-Response) ..... 41
3.2.12 PassiveTest_ICMP_Info (Stimulus-Response) ..... 43
3.2.13 PassiveTest_ICMP_TS (Stimulus-Response) ..... 44
3.2.14 PassiveTest_ICMP_Mask (Stimulus-Response) ..... 45

4. Collecting the signatures ..... 47
4.1 Computer Network Testbed ..... 47
4.2 Stimulation Procedures and Traffic Capture ..... 51
4.2.1 PassiveTest_TCP_SYN ..... 52
4.2.2 PassiveTest_TCP_ISN ..... 52
4.2.3 PassiveTest_IP_ID ..... 53
4.2.4 PassiveTest_TCP_TS ..... 53
4.2.5 PassiveTest_ICMP_ID_SEQ ..... 53
4.2.6 PassiveTest_ARP_Request and PassiveTest_ARP_Retransmit ..... 54
4.2.7 PassiveTest_TCP_SYNACK and PassiveTest_TCP_RSTACK ..... 54
4.2.8 PassiveTest_ICMP_Unreach/Echo/Info/TS/Mask ..... 55

5. Field Test Evaluation ..... 56

6. Development State of the Prototype ........ 70

7. Limitations and Future Work ........ 74

7.1 Limitations ........ 74

7.1.1 Fingerprinting countermeasures ........ 74

7.1.2 Network Conditions and Configuration ........ 75

7.2 Future Work ........ 76

8. Conclusion ........ 78

References ........ 85

Annex A: Collected Signatures ........ 88

Annex B: Active OS identification tools: Analysis of Nmap and Xprobe ........ 143

OS scan with Nmap ........ 143

Description of tests T1-T7 ........ 144

Description of the PU test ........ 146

Description of test Tseq ........ 148

General Remarks on OS differences ........ 151

OS scan with Xprobe ........ 152

Description of tests based on the UDP packet ........ 152

ICMP Echo Request packet ........ 153

ICMP Timestamp Request ........ 154

ICMP Information Request ........ 154

ICMP Address Mask Request ........ 154

General Remarks on OS differences ........ 154

Graphical representation of Xprobe's logic tree ........ 154

List of symbols/abbreviations/acronyms/initialisms ........ 160

Glossary ........ 161

## List of figures

Figure 1. Limitation due to the position of the monitor when doing sequencing analysis ....... 24

Figure 2. Testbed from which the signatures were collected.................................................. 48

Figure 3. Test selection menu ........................................................................................ 71

Figure 4. CRC' s Active Network Mapping Tool GUI............................................................ 72

Figure 5. CRC' s Passive Network Monitoring Tool GUI ...................................................... 73

## List of tables

Table 1. Typical Maximum Transmission Units (MTUs) ....................................................... 15

Table 2. Tested Operating Systems ........................................................................... 49

Table 3. Tested Operating system: Linux distribution............................................................ 50

Table 4. Stimuli used to collect SYN/ACK and RST/ACK signatures ................................... 54

Table 5. Traffic Trace #1 ................................................................................................ 56

Table 6. Traffic Trace #2 ................................................................................................ 57

Table 7. Results of individual tests.................................................................................. 59

Table 8. Tests Parameters .............................................................................................. 60

Table 9. Results obtained from Traffic Trace #1 for each host ............................................... 62

Table 10. Results obtained from Traffic Trace #2 for each host ............................................. 65

Table 11. Comments on each test .................................................................................... 80

Table 12. PassiveTest_TCP_SYN ..................................................................................... 88

Table 13. PassiveTest_ARPRequest.................................................................................. 89

Table 14. PassiveTest_TCP_ISN....................................................................................... 91

Table 15. PassiveTest_Echo_IP_ID (Subtest of PassiveTest_IP_ID) ..................................... 95

Table 16. PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) .................................... 95

Table 17. PassiveTest_IP_ID ..................................................................................... 100

Table 18. PassiveTest_TCP_TS ................................................................................. 104

Table 19. PassiveTest_ARPRetransmit ..................................................................... 105

Table 20. PassiveTest_TCP_SYNACK ..................................................................... 107

Table 21. PassiveTest_TCP_RSTACK ...................................................................... 127

Table 22. PassiveTest_ICMP_Unreach ..................................................................... 130

Table 23. PassiveTest_ICMP_Echo ........................................................................... 134

Table 24. PassiveTest_ICMP_Info ............................................................................. 135

Table 25. PassiveTest_ICMP_TS ............................................................................... 136

Table 26. PassiveTest_ICMP_Mask .......................................................................... 137

Table 27. PassiveTest_ICMP_ID_SEQ ..................................................................... 138

Table 28. PassiveTest_ICMP_ID (Subtest of PassiveTest_ICMP_ID_SEQ) ......................... 140

Table 29. PassiveTest_ICMP_SEQ (Subtest of PassiveTest_ICMP_ID_SEQ) ..................... 142

# Acknowledgements

This work was conducted at the Communications Research Centre Canada (CRC) and was partially funded by Defence R&D Canada (DRDC), National Defence. The author would like to thank her co-workers, Frédéric Massicotte, Denis Fournier, and Daniel Tremblay and also the co-op students who came to CRC during the years 2001/2002 for their tremendous contribution in the development of the prototype. Thanks to the people of the Information Networks and Systems (DINFO) team at CRC, in particular Peter Corrigan, for their help in collecting data for a field trial. The author would also like to thank Joanne Treurniet from DRDC for testing some of the techniques on a different data set. Thanks also to Tim Symchych and Frédéric Massicotte from CRC for their advice and support during the writing of this document.

# 1. Introduction

The ability to remotely identify a target operating system (OS) and version is a definite advantage when trying to identify vulnerabilities in networked systems. For this reason, there has been a lot of effort from the networking community to develop tools for OS detection, also referred to as OS fingerprinting. It is also an interesting area of research for network security analysts as it provides significant contextual information regarding the components connected to a network. Network administrators should be able to fingerprint machines under their control. OS discovery can help identify possibly vulnerable hosts connected to the network being protected. It can also help address the problem of false-positives from Network Intrusion Detection Systems (NIDS).

Traditional network security devices such as NIDS, firewalls, and security scanners usually operate independently of one another, with virtually no knowledge of the network assets they are defending. This lack of information can result in ambiguities when interpreting alerts and making decisions on adequate responses. Even with increased accuracy of security devices, network security analysts still must sort through a tremendous number of potential security events. OS identification can provide timely significant information about the components referenced by the alarm. This contextual knowledge can help reduce the rate of false positive alarms. Several attacks have distinctive signatures based on port numbers and data content and can thereby be detected by traditional NIDS. The NIDS are however likely to raise false alarms if an attack is randomly targeting several computers. Many of the targeted systems may not be vulnerable to this attack. To allow network administrators focus their attention on vulnerable targeted systems, an alert generated for a computer running a non-vulnerable OS should get a low priority. When attacks do occur or when a known virus is spreading, the OS information can be used to ensure that human resources are not wasted chasing down false positives.

Based on observations made during the analysis of several passive and active OS detection tools, we have developed a series of tests for detecting operating systems passively. Some of these are based on novel ideas; several others are adaptation of techniques that are normally conducted actively by other OS fingerprinting tools. We confine the analysis to headers at the data-link, network, and transport layers to avoid relying on access to application data. Tests are conducted on the headers of different kinds of protocols: ARP, IP, ICMP, UDP and TCP. The methodology goes beyond individual packet analysis. Stimulus and response packets are identified, paired, and are evaluated together to allow for more accuracy. Our method also allows for analyzing samples of packets transmitted by a computer (typically to observe how a certain header field evolves). We have developed a prototype based on these adapted techniques and built a database containing the fingerprints of almost 200 versions of operating systems.

The remainder of the document is structured as follows: the background section (section 2) covers the essential concepts of OS fingerprinting. It discusses other related works and contains a detail description of the fields that can be used for detecting OSes. Section 3 describes the three categories of passive tests developed (Singleton, Sample, Stimulus-Response). It then provides technical details for each of the fourteen individual tests. Section 4 describes the process used for collecting the signatures from the testbed network and section 5 describes preliminary results obtained on a corporate network.

# 2. Background

Fingerprinting methods can be broadly classified into two categories: active and passive. The term active refers to methods that inject traffic into the network. These methods typically consist of sending carefully chosen test packets and observing the reaction they stimulate. In contrast, passive techniques observe existing traffic without disturbing the networking environment, and are therefore less intrusive. Typically, passive methods consist of analysing traffic traces captured using a packet filter utility such as *tcpdump*[1]. Whether the methods are active or passive, OS fingerprinting is based on the principle that there are differences in the implementation of the network protocols (often called networking stacks) among various operating systems.

## 2.1 Active Fingerprinting Techniques

The differences among the networking stack implementations are especially noticeable when it comes to handling non standard (or abnormal) packets. To detect these differences, one can send a carefully crafted packet (a stimulus) to a target and analyse the behaviour it provoked. Note that the target's behaviour could be to not respond at all.

Some tools such as *ring*[2] or *Induce-ARP*[3] use techniques based on the retransmission behaviour of their target. *Ring* works at the TCP level and was developed by the Intranode Research Team. It stimulates its target with a SYN packet sent to an open port, when it receives a SYN/ACK response from its target, instead of completing the normal TCP three way handshake by sending an ACK packet back, it remains quiet. It is at this point that the targets will respond differently according to their TCP stack implementation. Some will give up on the communication right away, others, believing a packet loss has occurred, will retransmit the SYN/ACK packet. Since this packet will also remain unanswered, some targets will retransmit again, and so forth until they finally give up. *Ring* detects what operating system the target is running based on the number of retries and the delays between them. The current version has a limited number of signatures, but OSes do seem to be distinguishable according to this technique. For example, Windows 98 sends 3 retransmissions, waiting 3 seconds before sending the first retransmission, then 6 and 12 seconds between the other two. Windows 2000 behaves similarly but stops after the first two retransmissions. Not all OSes double the delays between retransmissions, and some send much more than three. The reader can refer to the document [2] describing *ring* for more details.

The principle behind *Induce-ARP* is similar but operates at the Address Resolution Protocol (ARP) level. The user of the tool must be connected to the same "ARP-utilising" link layer (e.g. Ethernet, FDDI) as its target. Computers connected to such networks can communicate with one another provided they have their peer's IP and hardware addresses. ARP requests are issued each time a host needs to obtain the hardware address (also known as the MAC address) associated with an IP address.

*Induce-ARP* stimulates its target using an ICMP echo Request with a spoofed[1], unused[2], source IP address. Upon reception of the forged packet, the target broadcasts an ARP request, querying for the MAC address associated with the IP address it thinks it was probed from. But because this IP address is unused, the ARP request remains unanswered. This leads to two situations, either the target gives up on this communication, or it reissues the ARP request. While most operating systems give up after the first attempt, some OSes such as Solaris and Linux can be fingerprinted using this technique, as they do send a few retransmissions. Solaris tries 6 times with a 1 second delay between each retransmission, while Linux tries 3 times waiting 1 second between each attempt. An attractive aspect of this technique is that ARP requests to nonexistent IP address are likely to be seen as part of regular traffic. For example, this happens when a user tries to communicate with a temporarily down machine, or when a user simply mistypes the IP address he or she is trying to reach. Note however that any identification based on delays may be unreliable due to network congestion.

Other tools go deeper in the packets they receive from their target to see how header fields have been set. This often leads to more precise OS detection. *Nmap*[4], a popular port-scanning tool, has been equipped with active OS detection capabilities for some times now. The tool is still being actively enhanced and the fingerprint database for its OS tests is quite impressive. It contains fingerprints for the most to the least popular OSes one can have access to, and the results often pinpoint the version quite precisely. The fingerprinting approach was inspired by an earlier tool named *queSO*[5]. One problem with *nmap* is that it generates a lot of traffic and some of its stimuli contain abnormal settings that can trigger alarms from intrusion detection systems (IDS). Nonetheless, reviewing the techniques used by the tool and their results is quite instructive. Fyodor, the author of *nmap*, has produced an introductory paper [4] on TCP/IP stack fingerprinting which first appeared in the Phrack e-zine (issue 54 article 9). Several passive tests developed during this project were inspired by *nmap*. For reference purposes, the techniques used by this tool are summarized in Annex B.

While *nmap* mainly focuses on differences in the TCP protocol implementations, another OS detection tool, *xprobe*[6], achieves good accuracy using the ICMP protocol. *Xprobe*'s project is fairly recent and the variety of tested operating systems is a little constrained. On the other hand, the number of packets required to fingerprint a target is smaller, and the packets used are likely to appear on a daily basis as part of regular network traffic. *Xprobe* is therefore stealthier. Its author, Ofir Arkin, has produce documents[6][7] on the techniques used and observations made regarding the differences among the ICMP protocol implementations. An overview of *xprobe* is provided in Annex B.

[1] "Spoofing" means forging the sender's identity so that the packet appears as coming from somebody else.

[2] By the term "unused", we mean that no host connected to the network at the time of probing is configured with this IP address. For the test to work, the spoofed IP address must be unused, and within the subnet range of the other two parties (the target, and the host running the tool).

## 2.2 Passive Fingerprinting Techniques

Instead of probing the target with crafted packets as active techniques do, passive techniques capture packets flowing on the network and inspect their content. The packet capturing process does not disturb the communications; it simply "listens" for them. Passive techniques typically listen for special kinds of packets and then inspect how header field values have been set. One must keep in mind that some parameters vary depending on the state of the connection. For example, the window size field in a TCP packet advertises the number of bytes the host is prepared to receive in its buffer space. While OSes tend to start with their own default value at the TCP connection set-up, the window size is very likely to change throughout a session as packets are transmitted and processed. This means that depending on the fields examined, not all packets of a connection may be suitable for analysis.

In the summer of 1999, a person with the online name "photon" posted a message to the nmap-hackers mailing list [8] in which he described some ideas for doing passive OS fingerprinting. The thought of doing OS detection without disturbing communications, or from another point of view, without being detected, is very attractive and several lines of research have been active since then.

For example, Michal Zalewski has developed a tool called *p0f*[9] that was initially listening for SYN packets (the tool has since evolved to examined a few other types of packets). Another tool, which includes a passive OS detection functionality along with several other network mapping capabilities, is *ettercap*[10]. When running in passive mode, *ettercap* captures either SYN or SYN/ACK packets to identify the OS. While a SYN and a SYN/ACK packet are related, they are inspected by ettercap independently. Some care should be taken when interpreting the results *ettercap* and *p0f* produce based on SYN/ACK packets, as some of the fields these tools examine depend on what had been sent in the SYN packet. To the best of our knowledge, no currently available tools take this into account.

Jose Nazario[11] from Crime Labs Research has written a paper on passive fingerprinting using network client applications. The technique he describes consists of looking at the application layer, seeking special strings that could identify the operating system. Telnet and FTP banners for instance often state in clear text the OS that the server is running. Some applications also involve option negotiation prior to exchanging any data, and because applications are often platform dependent, this can sometimes be used in OS fingerprinting. While information obtained in this manner can be quite precise[3], these techniques rely on the availability of the data at the application layer. This limits the applicability of the techniques. For example, the

[3] Masquerading can however be relatively easy at the applications layer using third party tools or simply by modifying the properties. Fields like HTTP User-Agent and X-Mailer are not even mandatory. The content of such fields can be overwritten without affecting the communication. In the same train of thought, "Banners" are systematically rewritten nowadays by system administrators. Obfuscation and masquerading at the network and transport layers are also feasible, but care must be taken not to break connectivity.

application layer may be encrypted, as it is the case with the Secure Shell (SSH) and the Secure Sockets Layer (SSL) protocols. Another limitation of an application layer packet analysis approach is when the application data in each packet is removed at the time of capture. Depending on the organisation's policy, this may be done for privacy issues[4]. Limiting the capture length of packet is also a common practice when storage capacity is in place to allow for post analysis of traffic traces. Nonetheless, the techniques presented in [11] are simple to implement and can produce reasonably accurate results.

## 2.3 Header fields used in fingerprinting

This section enumerates the primary fields contained in IP, TCP, ARP, and ICMP headers that can be used for detecting OSes. These protocols belong to the IPv4 protocol suite, which is the focus of this work. For each field, we give a brief description of its intended purpose, and then we describe how it can be used in OS fingerprinting.

### 2.3.1 IP Don't Fragment bit

Setting the IP "don't Fragment" (DF) to 1 instead of 0 specifies that the IP datagram should not be fragmented. Many operating systems set the DF bit by default in some of the packets they send. For example, a lot of systems have this bit set in datagrams carrying TCP segments with the SYN flag on. However, even then, there are some differences: before setting the DF bit, some implementations ensure that "only" the SYN flag is set, while some others check that "at least" the SYN flag is set. This means that some systems have the DF bit set in SYN but not in SYN/ACK packets, and some others have it set in both SYN and SYN/ACK packets. There are several other kinds of packets for which the DF bit is set differently by OSes.

### 2.3.2 IP Time-To-Live

The IP "Time-To-Live" (TTL) field sets an upper limit on the number of routers a packet can pass through. This prevents a packet from getting caught in routing loops. The value is initialised by the sender, and decremented by one by every router that processes the packet. The initial value varies depending on the Operating system. Moreover, some OSes use different values depending on the type of packet they send. For example BSD-like systems use a value of 64 in a datagram carrying a TCP segment, and a value of 255 in an ICMP message. From experiments conducted on our testbed, we were also able to observe that, in special cases, some OSes echo the TTL value they receive instead of using their own default value. This happens with some Solaris, Mac OS, and Novell versions for instance.

[4] It is at this layer, for example, that one finds login names, passwords, or e-mail messages.

### 2.3.3 IP Service Type

The 8-bit long IP Service Type as described in RFC 1349[12] consists of three fields. The first field, “Precedence” is 3-bit long and is intended to prioritise the IP datagram. The second field, “Type-of-Service” (TOS), is 4-bit long and is intended to describe how the network should make tradeoffs between throughput, delay, reliability, and monetary cost. The last field, “Must-Be-Zero” (MBZ), is only one bit long and is unused. The settings of the TOS bits can be chosen at the application level. Therefore, if the specifics of the application layer protocol are not considered, relying on the TOS field when doing OS fingerprinting can be misleading. The *xprobe* OS fingerprinting tool takes the TOS field into account because some systems set the Precedence bits to a special default value when sending an ICMP error message.

The whole octet is now being replaced by the Differentiated Services mechanism for Quality of Service. As specified by RFC 2474[13] and RFC 3168[14], it consists of two fields: the first 6 bits make the “Differentiated Services Codepoint” (DSCP), and the last two bits describe the “Explicit Congestion Notification” (ECN) field at the IP level. The key point is that with Differentiated Services mechanisms, this octet is no longer the sole concern of the two end-points only: intermediate routers handling the packet may change the setting. This limits even more the reliability of this field when fingerprinting end-point systems.

### 2.3.4 IP Identification

The IP “Identification” (ID) field uniquely identifies each IP datagram sent by a host. It plays an important role in the reassembly of fragmented datagrams. A guideline in RFC 791[15] indicates that the upper layer that is having the IP layer send the datagram should choose the value. This implies that two different IP datagrams, one carrying TCP and one carrying UDP, can have the same identification field. Note that while this doesn’t cause any reassembly problem, most operating systems have the IP layer increment a kernel variable each time an IP datagram is sent, regardless of the upper layer. Therefore, in most cases the IP identification field is incremented by one each time the system sends a new datagram. Linux kernels 2.4.x are counter-examples to this simple incremental behaviour. In addition to zeroing out the IP ID value of some special packets, these systems maintain separate counters for different connections. Solaris and Mac OS prior to Mac OS X also use separate counters, one per destination address (independently from the protocol and the port numbers). Recent versions of OpenBSD (2.5 and above) use a pseudo-random generator for the IP ID of each IP datagram. Some other systems such as Windows 95 have a monotonically-increasing behaviour, but instead of incrementing the field by 1 (or 0x0001 in hexadecimal), they increment it by 256 (or 0x0100 in hexadecimal). This is due to a byte order mishandling (they do not bother putting the counter into network (big-endian) byte ordering. For example, the IP ID following 0x1234 will be 0x1334, not 0x1235. *Nmap* is the first tool we have seen that analyses the IP ID incremental behaviour of its target when doing OS fingerprinting. It does not have, however, a category for OSes that use session dependent counters.

### 2.3.5 ARP Target Hardware address

The ARP protocol allows a host to ask for the physical address of another host connected to the same physical network, given only the IP address of this other host. The headers of an ARP request and an ARP reply have the same format. Among other fields, there is one (named *Operation*) that identifies whether it is a Request or a Reply, and four fields to bind IP addresses to Physical addresses. These four fields are the *Source Hardware address*, the *Source IP address*, *the Target Hardware address*, and the *Target IP address*. When a host sends a request, it fills the *Source Hardware address* and the *Source IP address* with his own, and also supplies the *Target IP address* for which it is requesting the physical address. Before the target replies, it fills the missing address (*Target Hardware address*), swaps the target and sender pairs of addresses, and changes the operation code to “reply”.

We observed that when sending the request, the content of the “blank field” (*Target Hardware address*) varies with operating systems. Some initialise it with 0x000000000000, others fill it with 0xffffffffffff. Moreover, some versions of FreeBSD forgot to initialise the field and so it contains allocated memory garbage.

### 2.3.6 ICMP code

The Internet Control Message Protocol (ICMP) assists TCP/IP communications by providing a mechanism to handle errors and control messages. ICMP headers have a *code* field that accompanies the *type* field. Together, they describe the purpose of an ICMP message. For example, an ICMP message with type 3, code 2, is a Destination Protocol Unreachable message. If type is still 3 but code is 3, it is a Destination Port Unreachable message. For some types of ICMP messages, the code field is meaningless (the purpose of the message is defined by the *type* field on its own). This is the case for ICMP Echo messages that are used to determine whether a machine is alive (i.e. reachable and responding) on the network. The ICMP Echo request message is of type 8, and the response (ICMP Echo Reply) is of type 0. While RFC 792[16] guidelines are to set the *code* value to zero in both cases (the request and the response), the construction of the Echo Reply message is described as follows:

> “To form an echo reply message, the source and destination addresses are simply reversed, the type field changed to 0, and the checksum recomputed.”

That is, the RFC does not mention how to handle the *code* field.

Windows family systems can be detected using an active test that sends one ICMP echo Request. It suffices to set the code field to a nonzero value. In the response, Windows systems overwrite this value to set the field to zero, while other systems simply echo the value contained in the Request. This fingerprint technique is used by *xprobe*. We also noticed from experiments in the testbed that some OSes do not respond to ICMP echo Request when the ICMP code of is nonzero.

### 2.3.7 ICMP Identifier and ICMP Sequence Number

ICMP Echo Request/Reply messages contain two header fields to aid in matching the replies with the requests. They are named *Identifier* and *Sequence Number*. As quoted from RFC 792:

> “The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.”

The *ping* utility is included with most platforms to allow testing for TCP/IP connectivity. It uses Echo Requests datagram to elicit ICMP Echo Responses from a host or gateway. There exist differences among ping implementations in the setting of the ICMP Identifier and ICMP Sequence Number parameters. Most implementations use the *Identifier* to identify Echo Requests aimed at different destination addresses, and use the *Sequence Number* to identify Echo Requests when multiple requests are sent to the same destination address. This generally leads to the following behaviour: When the *ping* application sends *n* echo requests to a given destination address, the ICMP *Identifier* is fixed to a certain value and the *Sequence Number* is incremented *n* times, starting at zero. Next time *ping* is called, it will choose a new ICMP *Identifier* and reset the *Sequence Number* to zero. *Ping* utilities based on this behaviour still show some differences depending on the flavour. The ICMP *Sequence Number* is sometimes incremented by 0x0001, and sometimes by 0x0100. Some use the Process ID (PID) as their ICMP *Identifier* for each call of the application.

The *ping* utility in Windows systems has a different behaviour than the one described above. They use a constant value for the ICMP *Identifier* in all of the ICMP Echo requests, and a global counter for the ICMP *Sequence Number*. For example, the constant ICMP *Identifier* value is 0x0200 for Windows 98/2000/XP, 0x0100 for Windows 95/NT, and 0x0300 for Windows Me. The Windows ICMP *Sequence Number* counter is global in the sense that it starts at 0x0100 when the first ICMP Echo Request is issued after a reboot (or when the value reaches its maximum 16-bits value), and is then incremented by 0x0100 for any subsequent ICMP Echo Request sent. While [7] describes observed behaviour of the ICMP *Identifier* and the ICMP *Sequence Number*, we have not seen any OS fingerprinting tool that listens for ICMP Echo Requests to infer the OS based on how these two fields are set.

### 2.3.8 Data of ICMP Echo messages

Adding data into the ICMP Echo Requests allows detection of data-dependent transmission problems that may occur along the path. RFC 792 specifies that the data received in the echo request must be returned in the echo reply message. It does not specify how much data should be sent, or what the data should be. Most *ping* utilities allow the user to choose a pattern to transmit. If the option is not used, the *ping* utility sends its own default data. This data typically includes a fixed portion identical in all Echo Requests the *ping* utility transmits. Windows's *ping* utility sends 32 bytes of data identical in each echo requests. The Unix-based *ping* utility sends 56 bytes of data, the last 48 bytes are fixed and the first 8 bytes consist of a timestamp used for the calculation of the round trip (see the man page of *ping* for more details). Novell's *ping* utility sends 12 bytes, of which only the last two are fixed. By examining the length and content of the ICMP data, one can define a test that passively tries to identify the *ping* utility used, which in turn can be associated with a given OS.

### 2.3.9 Data of ICMP Error messages

There are several types of ICMP messages that are defined to report an error in the processing of a datagram. Examples of such ICMP error messages are Destination Unreachable (ICMP type 3), Source Quench (ICMP type 4), Redirect (ICMP type 5), Time Exceeded (ICMP type 11), and Parameter problem (ICMP type 12). These messages are delivered to the sender of the packet that generated the error. They carry the IP header and at least the first 64 bits of the next higher header. This is done to help the host that sent the offending packet to match the error with the appropriate process. If the protocol above IP uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data. Most of the Operating systems will send back the IP header and the first 64 bits of the next header only. Others will send more. There are also several OSes that alter the datagram echoed in the ICMP error message. When this happens, it means that there will be discrepancies between certain immutable fields[5] in the original datagram and those of the echoed version. Both *nmap* and *xprobe* exploit these particularities to fingerprint their targets by probing an ICMP Port Unreachable Message (they send a UDP packet to a closed port).

### 2.3.10 TCP Sequence Number

If we consider the stream of bytes flowing in one direction, the TCP protocol numbers each byte with a sequence number. The TCP “Sequence number” field identifies the sequence number of the first byte carried in the packets. When a new connection is established, the Sequence Number field contains the Initial Sequence Number (ISN) chosen by the host for this particular connection. The way the ISN is chosen when establishing a new TCP connection varies with Operating systems. To the best of our knowledge, *nmap* is the first tool that tries to deduce how the ISN numbers are generated in order to identify the Operating system.

### 2.3.11 TCP Acknowledgment Number

As mentioned above, every byte of a TCP data stream is numbered. The “Acknowledgment Number” field gives the next sequence number that the sender of the acknowledgment expects to receive. Generally, this is equal to the Sequence Number of the last successfully received byte of data, plus 1. This field is valid only when the ACK flag is on. When responding to a TCP segment having abnormal settings, some TCP implementation set the Acknowledgment Number value differently. Both *queSO* and *nmap* verify the value of this field in the responses they get.

### 2.3.12 TCP Flags

Some TCP segments carry only an acknowledgment, while some others also carry data. Some segments are requests to initiate or to terminate a connection. There are 6 flag bits in the TCP header indicating the “purpose” of a segment. One or more of them can be turned on at the same time. Manipulation of TCP flags has been a focal point in many OS fingerprint

[5] The immutable fields are those that do not change in transit. This is in contrast with the TTL and the checksum fields that do change as they are being processed by the routers along the way.

experiments. When an unconventional setting is used, it can put the TCP stack of the target in an “undefined state”, and thus the reaction may differ depending of the OS. *QueSO* and *nmap* use this technique in several of their tests. Following is a brief description of each TCP flag:

### URG

The “Urgent Pointer” flag is used to tell the other end that "urgent data" of some sort has been placed into the stream of data. When the flag is set to 1, it validates the value of the “Urgent pointer” field of the TCP header. When it is set to 0, the “Urgent pointer” field is meaningless and thus typically zeroed out. The author of *p0f* mentions that some Windows systems do not always zero out the “urgent pointer” field although the URG flag equals 0 [9]. We cannot confirm this allegation since this behaviour has not been seen for any TCP packets produced by the operating systems tested in the lab. That is, all TCP packets with URG flag equalled to 0 that we examined also had the “Urgent pointer” field set to 0.

### ACK

The “Acknowledgment” flag indicates that the reception of data is being acknowledged, and it validates the value of the “Acknowledgment Number” field.

### PSH

The “Push” flag tells the TCP stack of the receiver to pass the data to the Application layer as soon as possible. This data would consist of whatever is in the segment with the PUSH flag, along with any other data the receiving TCP has collected and buffered for the receiving process. It is useful for an interactive application for example. When a client sends a command to a server, the client expects its command to be processed rather than to remain in the TCP buffer waiting for additional data.

### RST

The “Reset” flag informs the other side that a connection problem has occurred. In general, TCP sends a RST after handling a segment that doesn't appear correct for the referenced connection (connection specified by the quadruplet: source IP address, destination IP address, source port, and destination port).

#### SYN

The SYN flag is sent in segments that initiate a connection in order to synchronize the Initial Sequence Numbers each side is starting with. The SYN flag appears in the first two packets of a connection. Suppose a communication is about to be established between hosts **A** and **B**, and that **A** is the "caller". To initiate the connection, **A** sends a SYN packet containing an Initial Sequence Number (ISN) and in which the only flag set is SYN. **B** then responds with a SYN/ACK packet (in which both SYN and ACK are set) informing **A** of its own ISN, and acknowledging **A**'s ISN. These are the first two steps of the "Three Way Handshake" that needs to be completed before any data is exchange with TCP. The third and last step that completes the Three Way Handshake is host **A** sending an ACK to host **B** in order to acknowledge the ISN of **B**.

#### FIN

The sender of a FIN flag indicates it has finished transmitting data. A connection typically terminates after both sides have sent a FIN and have acknowledged the reception of the FIN coming from the other side.

### 2.3.13 TCP Reserved

As specified by RFC 793, the TCP header contains a 6-bit field reserved for future use. RFC 3168[14] now describes how the last two bits of this field can be used for control of congestion (Explicit Congestion Notification). The two special bits are referred to as "Congestion Window Reduced" (CWR) and ECN-Echo (ECE) respectively. They are set by the endpoints of a connection to signify that the endpoints are ECN capable. As proposed by RFC 3168, it is during the TCP connection set-up phase that the source and destination informs one another about their desire and/or capability to participate in Explicit Congestion Notification (ECN). The expected behaviour when both parties support the ECN capability is the following: the initiator of the connection turns on the ECN and CWR flags in the SYN packet, and the receiver responds by setting the ECN flag (but not the CWR flag) in the SYN/ACK packet. While the ECN capability had been suggested in the past, it became a proposed standard only recently[6], and thus several OSes do not support this capability by default. Both *queSO* and *nmap* use the ECN bits in one of their tests (the one that sends a SYN packet to an open port). While *queSO*'s usage complies with the RFC, *nmap*'s does not. In either case, the technique is useful for recognising some OSes.

[6] The issue date of the *proposed standard* RFC 3168 is September 2001. The ECN mechanism was introduced in RFC 2481 (1999) under the *experimental* status.

### 2.3.14 TCP Window

The TCP "Window" field is used to advertise how many additional bytes of data the sender of the packet is prepared to accept. This "receive window space" can be thought as the currently available buffer size. Throughout the lifetime of the connection, each endpoint informs the other side of its current value. The value is very likely to change during the connection as data is received. The default values of socket buffer size available at the beginning of a connection widely differ between implementations. Older Berkeley-derived implementations would set a default value to 4KB, but newer systems use larger values (up to 64KB). These default values can be seen in SYN and SYN/ACK packets and are widely used in fingerprinting techniques, whether they are active or passive. For some OSes, the value in these packets is a multiple of the advertised Maximum Segment Size (MSS) found in the TCP options (see section 2.3.15).

### 2.3.15 TCP Options

While IP options are rarely used, TCP options are seen quite frequently. The space they occupy at the end of the TCP header is of variable length. When there is no TCP option, the TCP header is exactly 20-byte long. The length of each option is a multiple of 8 bits (1 byte), and some are as long as 10 bytes. Not all OSes support the same TCP options, nor do they advertise them in the same order. Moreover, the values these options take may differ depending on the operating system. Depending on their purposes and definitions, some options must only be used with special segments, for example when the SYN flag is set.

Passive OS detection tools often look at options set in a SYN packet, while most active tools look at those set in a SYN/ACK packet in response to their stimulus. The question "What options are supported?" is best answered in the latter case. This is because some OS support a lot of options, but ask for few. The rule of thumb is that when a host is queried with a set of options, it usually shows support of the options it can handle by setting them in the reply. We describe below some commonly used TCP options.

#### Maximum Segment Size (MSS)

TCP uses the "Maximum Segment Size" (MSS) option to inform the other side of the Maximum Transfer Unit (MTU) on his side. Basically, when initiating the connection, both ends will announce the maximum IP datagram size that can pass through the link layer they are connected to, without being fragmented. If connected to an Ethernet cable for instance, the MTU is 1500. Table 1, borrowed from *TCP/IP Illustrated* [17], provides a list of typical underlying technologies with the corresponding MTU.

*Table 1. Typical Maximum Transmission Units (MTUs)*

| Network | MTU (bytes) |
|---|---|
| Hyperchannel | 65535 |
| 16 Mbits/sec token ring (IBM) | 17914 |
| 4 Mbits/sec token ring (IEEE 802.5) | 4464 |
| FDDI | 4352 |
| Ethernet | 1500 |
| IEEE 802.3/802.2 | 1492 |
| X.25 | 576 |
| Point-to-Point (low delay) | 296 |

The MSS is announced in SYN and SYN/ACK packets, and should only bee seen in those. The MSS is the largest data segment that can be carried in the IP datagram. It is equal to the MTU minus the length of the IP and TCP headers. Operating systems will typically announce a MSS equal to the MTU minus 40 (MTU – 20 bytes of IP header – 20 bytes of TCP header), no matter how many options are being advertised.

While the MSS very much depends on the underlying technology to which a host is connected, some OSes calculate the value in peculiar ways. For instance *nmap* uses the fact that when queried with a very low MSS value in a SYN packet, old Linux kernels would respond by echoing the value in their SYN/ACK instead of stating the actual MSS possible at their end. During this study, we observed other particular behaviours. OpenBSD versions 2.5-2.7 for instance advertise a shorter MSS in a SYN/ACK segment containing the TCP Timestamp option. This is because at this point of the connection set-up, these systems are aware that the TCP timestamp option will be used throughout the session, and will consume space in the TCP headers of subsequent segments[7]. OpenBSD version 2.8 and above still take into account the space needed for options when calculating the MSS, but advertise a value equals to MTU-40 whether the segment is a SYN or a SYN/ACK. Another OS, QNX RTP 6.0, also demonstrates a particular MSS setting. The value it advertises in both SYN and SYN/ACK segments is equal to MTU-41. This was fixed in later QNX versions.

### No Operation (NOP)

The No Operation (NOP) TCP option is used to provide padding around other options, for example, to align the beginning of the next option on a 32-bit word boundary. Unlike the MSS, the NOP option may appear in any TCP segment. The use of this option is not mandatory, and it is explicitly

[7] See implementation details from source code of OpenBSD 2.5-2.7 file /netinet/tcp_input.c.

specified in RFC 793[18] that the receiver must be prepared to process options even if they do not begin on a 32-bit word boundary. Operating systems tend to use it differently. When looking at the order in which the TCP options appear, *nmap* takes into account the location of the NOPs.

### End of Option List (EOL)

As its name indicates, The End of Option List (EOL) signifies that the end of the TCP options have been reached (i.e. there are no more TCP options to follow). Note that the list of options may be shorter than the Data Offset field might imply. This is because the Data Offset field is given in 32-bit words, but the length of the TCP options might not be a multiple of 32 bits. The EOL option needs only to be used if the end of the options would not otherwise coincide with the end of the TCP header. Some OSes will use NOP between options instead of using the EOL at the end. Macintosh systems are among the rare OSes to use the EOL option. *Nmap* was the first OS detection tool to check for the EOL TCP option. *Ettercap*, and since recently *p0f*, perform that check also.

### Window Scale (WSCALE)

The Window field described earlier is only 16 bits long. This limits the maximum window size to 65535 bytes. The “Window Scale” option was defined to allow a host to advertise a buffer space bigger than 65535 bytes. As described in RFC 1323, the option has two purposes: (1) indicate that the TCP is prepared to both send and receive window scaling, and (2) communicate a scale factor to be applied to its receive window. To enable window scaling in either direction, both sides must send Window Scale options in their SYN segments. Like the MSS, the WSCALE option should only appear in SYN and SYN/ACK packets. It should not be seen in a SYN/ACK packet if it was not first advertised in the SYN packet.

The Window Scale value gives the number of bits by which the Window size field’s value should be shitted when expressed in binary. For example, suppose two Operating Systems advertise a Window of 65535 (1111111111111111 in binary), but one had set a Window Scale value of 1 in its SYN segment, and the other a Window Scale value of 2. The former is announcing a buffer space of 65535x$2^1$ (11111111111111110 in binary), and the latter is announcing 65535x$2^2$ (111111111111111100 in binary). The Window Scale option is relatively new compared to the MSS option, and not all OSes implement it. Some OS detection tools, such as *nmap*, check to see whether or not the option is supported. The *p0f* tool is the only one we have seen that also considers the value of the Window Scale option.

## Timestamp

Also described in RFC 1323, is the Timestamp option. The purpose of this option is to estimate the Round Trip Time (RTT) in order to identify changes in latency, and thus identify situations that may require acknowledgment timer adjustments. The Timestamps option has two fields. The first field is the Timestamp Value (TSval), which contains the current value of the timestamp clock of the TCP sending the option. The second field is the Timestamp Echo Reply field (TSecr) and is only valid if the ACK bit is set in the TCP header. When valid, it echoes a TSval received from the remote TCP. The TSecr value will generally be from the most recent Timestamp option that was received. When TSecr is not valid, its value must be zero. Unlike the MSS and WSCALE options, the timestamp option is typically used throughout the TCP session.

The TSval is obtained from a (virtual) clock called the "timestamp clock". Its value must be at least approximately proportional to real time. The rate at which each system increments the clock varies between OSes. For example most BSD systems update the clock once every 500ms, while Linux systems update it more frequently (once every 10ms, or even once every 1ms for some kernels).

Another example of differences among OSes can be seen in Windows 2000/Me/XP. The update rate is once every 100ms, but they use the Timestamp option in a peculiar way. They support the option but won't advertise it when initiating a connection (i.e., when sending a segment containing a SYN bit and no ACK bit). When probed by a SYN packet having the Timestamp option set, they will acknowledge the option in their SYN/ACK, but with a TSval equal to 0. They wait until the Three Way Handshake is completed before sending their first nonzero TSval.

*Nmap* uses these existing differences regarding the TSval settings when guessing the operating system of a target. To do so, the tool makes multiple connections to its target and computes the update rate based on the elapsed time versus the TSval increment.

It is also possible to gain other pieces of information about a system by looking at the TSval. As pointed in [19], the TSval is, for some OSes, tied to the system uptime. For such systems, once the update rate of their timestamp clock is known, one can deduce the last time the computers have been rebooted by capturing a packet having the TCP Timestamp option set. While detecting the system uptime has nothing to do with OS detection, it can be useful information when monitoring a network.

### Selective Acknowledgments Permitted (SackOK) and Selective Acknowledgment Data (Sack)

The concept of selective acknowledgment is described in RFC 2018[20]. Its purpose is to allow a receiver to acknowledge non-consecutive data. When this mechanism is not used, a TCP receiver can only acknowledge the packets up to the Sequence Number immediately before a missing packet. This means for example that if 100 packets are received but the second packet is missing, the receiver can only acknowledge the receipt of the data contained in the 1st packet, so the sender would have to retransmit packets 2 through 100. By using Selective Acknowledgment, the receiver can acknowledge the receipt of packet one and all packets between 3 and 100. Thus, the sender only needs to retransmit packet 2. There are two options for the Selective Acknowledgment mechanism. The first is an enabling option, Selective Acknowledgment Permitted (SackOK), which may be sent in a SYN segment to indicate that the mechanism can be used once the connection is established. SackOK must be included in the TCP options in both the SYN and SYN/ACK packets of the Three Way Handshake, or it cannot be used. The other is the Selective Acknowledgment (Sack) option itself, which may be sent over an established connection once permitted by the SackOK notifications. This option is of variable length and gives a list of pairs of Sequence Numbers, where each pair defines a range of numbered bytes that are being acknowledged.

Selective Acknowledgment is only supported by a few Operating Systems (generally the most recent ones). Some OS detection tools, such as *nmap* or *p0f*, check whether or not the option is supported.

# 3. Passives Tests Developed

The network security research team at CRC has developed novel techniques for passive operating system discovery. Some of the techniques were inspired by active tools and adapted to be conducted passively on regular traffic. We have developed about a dozen tests that analyse headers of packets captured from a network. They are passive and thus they do not send any probe packets. Moreover, they only analyse headers at the link, network, and transport layer, thus the approach also has the advantage of remaining applicable whether the application layer is encrypted or not.

Our tests are conducted on the headers of various types of protocol: ARP, IP, ICMP, UDP and TCP. We have developed a prototype based on these techniques and built a database containing the fingerprints of close to 200 versions of operating system.

The passive techniques we use go beyond individual packet analysis. Stimulus and response packets are passively collected, identified, paired, and analysed together to allow for more accuracy. While matching corresponding stimulus and response is easily done with our approach, it seems to be overlooked by current available passive tools. Our method also allows for analysing samples of packets transmitted by a computer (typically to observe how a certain header field evolves). We define three categories of tests as described below.

## 3.1 Categories of Passive Tests

Through out this document, the term *test* refers to a series of specific criteria that are used to examine a given packet or group of packets. Below are the categories of tests we define.

### 3.1.1 Singleton

Tests in this category are conducted on a single packet (a singleton). They typically look for default values of header fields in order to identify the OS of the sender of the packet.

The general algorithm to perform a *Singleton* test is as follows:

1. Monitor traffic, listening for packets satisfying a certain filter;

2. Compute the signature once a packet is captured;

3. Obtain the corresponding OS (or group of OSes) from the signature database.

### 3.1.2 Sample

This second category of test requires capturing a sample of packets a host sends out. A *Sample* test typically analyses how a certain field evolves as consecutive packets are being transmitted. Any of our *Sample* tests requires two parameters: (1) the number of packets composing the sample, and (2) the maximum delay (in millisecond) that can separate the last packet from the first.

The general algorithm is this case is:

1. Monitor traffic, listening for packets satisfying a certain filter;

2. Hold in memory captured packets by Source IP address until a sample is complete.

3. Compute the signature on the sample;

4. Obtain the corresponding OS (or group of OSes) from the signature database.

As it will be seen shortly, the tests we have defined in this category may produce inaccurate results for several reasons, in particular if packets of a given sequence are missed, or if a machine under observation reboots during the sampling process. The maximum delay restriction is an attempt to limit the chances of getting samples that will produce inaccurate results.

### 3.1.3 Stimulus-Response

This category is required because some packets are “answers” to other previously transmitted packets. The settings in a response packet may depend on the request that was made. For example, the TCP options of a SYN/ACK packet are partly dependent on the TCP options of the SYN packet. It is the OS of the “responder” that such a test tries to identify (e.g. the sender of the SYN/ACK packet in the previous example). Note that this category of tests includes cases where the response from the target to a certain stimulus is to remain quiet. In these particular cases however, it is not always easy to passively determine whether the target system does not respond to such stimuli or whether it is simply down or even whether the response was missed because of asymmetric routing.

The general algorithm of a *Stimulus-Response* test is as follows:

1. Monitor traffic, listening for packets satisfying either the stimulus or the response filter;

2. If the captured packet is a stimulus, hold it in memory by destination IP address and go back to monitoring. If the packet is a response, search allocated memory for the corresponding stimulus and then go to step 3;

3. Compute the signature based on the stimulus-response pair;

4. Obtain the corresponding OS (or group of OSes) from the signature database to determine the OS of the sender of the response.

Stimulus-Response tests have a parameter that defines the maximum delay (in millisecond) an unanswered stimulus can remain in memory. Once the timeout is passed, unanswered stimuli may be processed depending on the particular test.

## 3.2 Tests Descriptions

### 3.2.1 PassiveTest_TCP_SYN (Singleton)

This test is conducted on the first packet that establishes a TCP connection (i.e. a SYN packet). It was inspired by *p0f*'s v1, and we have added a check for all TCP options, taking their order into account. The SYN test of *p0f* v2 has evolved to also take the order of TCP options into account.

Fields under analysis are found in the IP and TCP headers. The criteria we define are:

1. Is the IP Don't Fragment bit set in the IP header? ("Y" or "N")

2. What is the value of the IP Time To Live field? (one-byte integer expressed in decimal format)

3. What is the value of the TCP Window size field? (two-byte integer expressed in decimal format, or n(MSS) if Window size is "n" times the advertised MSS)

4. What TCP ECN bits, if any, are set? ("C"=CWR, "E"=Ecn-Echo)

5. What TCP options are advertised, and in what order do they appear? ("M"= Maximum Segment Size, "N"= No-operation, "T"= Timestamp, "W"= Window Scale, "S"= SackOK, "L"= End Of List, "C.New"= Connection Counts New[8]). We also capture the value for the Maximum Segment Size and Window Scale options using the format

[8] Connection Counts New (CC.NEW) is non-standard. It was defined in RFC 1644 in 1994, T/TCP -- TCP Extensions for Transactions Functional Specification, which has the RFC experimental status. This option is supported by old FreeBSD versions.

“@DecimalValue”. Similarly, if the tsval value of the timestamp option is zero, we denote this by T@0.

Signature example:
DF=Y;TTL=128;WIN=44(MSS);TCP_Ecn=;TCPopts=M@1460NNS;

Note that the TTL value decrements by one each time a packet passes through a router and the value of the Maximum Segment Size option depends not only on the OS but also on the MTU at the target end-point. The signatures in the database contain TTL and MSS values for local hosts sharing an Ethernet link. When the prototype program attempts to identify the OS of a remote host based on those signatures, special look-up algorithms are called upon to try to identify the most plausible matches for the TTL and the MSS.

The TCP Window size field and the value of the Window Scale TCP option of some OSes may vary depending on the application involved. As described later in section 4.2.1, the signatures were produced using a limited number of application clients. The signatures provided in Table 12 of Annex A are therefore not exhaustive. It can be observed from Table 12 that the order in which the TCP options are set helps distinguishing between the different operating systems family, and that the Window Size helps discriminate between versions among a given family.

### 3.2.2 PassiveTest_ARP_Request (Singleton)

This test is of type *Singleton*; the packet it listens for is an ARP request. It was developed based on Address Resolution Protocol (ARP) traffic observed within the network testbed. As discussed in section 2.3.5, the content of the *Target Hardware address* of an ARP request varies with operating systems. Most systems initialise it with 0x000000000000; others such as Solaris and the original Mac OS fill it with 0xffffffffffff. Some versions of FreeBSD even fail to initialise the field and so it contains allocated memory garbage.

Based on traffic observed on a network separate from the testbed, we have planned for two other categories of behaviours: systems that initialise the *Target Hardware address* with the value of their own *Source Hardware Address*, and systems that initialise it with the *Destination address* included in the Ethernet header when sending a directed ARP request (i.e. not broadcasted). This happens for systems that refresh their cache by asking the remote machine if it's still using that IP.

Note that because ARP is confined to the broadcasting environment on which the hosts are connected, the monitor will only see the ARP traffic from hosts on the same physical network.

The one field examined in this test appears in the ARP header. The criterion for differentiating between operating systems is the following:

1. What is the value of the *Target Hardware address* field? (“0x000000000000”, “0xffffffffffff”, “uninitialized”, “SourceHardwareAddress”, “EthernetDestinationAddress”)

Signature example: TargetHardwareAddress=0xffffffffffff;

The signatures collected appear in Table 13 of Annex A. It can be observed that FreeBSD 5.0 and 5.1 can easily be detected. Mac OS prior to Mac OS X and Solaris systems can also be distinguished from the other OS families based on this test.

### 3.2.3 PassiveTest_TCP_ISN (Sample)

This test tries to categorize the target based on Initial Sequence Numbers (ISNs) generation. Recall that the ISNs are exchanged during the TCP connection set-up; they correspond to the Sequence Numbers found in the SYN and SYN/ACK packets (see section 2.3.10 and 2.3.12).

Categorizing the ISNs generation requires the monitor to collect a sample of ISNs generated by the target. The ISNs sampled are then analysed to determine if and how they are related to one another. There may be uncertainties in the outcome of such a test for several reasons. As described below, some are related to the sampling technique, while others are related to the difficulty of defining robust classification algorithms.

First, depending on the network conditions and its configuration, there can be cases where the ISNs sampled were not generated one after the other. For example, the sample may contain duplicate ISNs when TCP retransmits packets due to packet loss during network congestion, or the order of the ISNs in the sample may differ from the order in which they were generated if packets arrive out of order to the monitor capturing the packets. It can also be that, due to its location, the monitor does not see all traffic emerging from the target. For example, suppose the network configuration depicted in Figure 1, where the cloud represent a switching environment that separates the shared media to which hosts **A** and **B** are connected from the one shared by hosts **C** and **D**. Suppose the system under observation (host **A**) is communicating with hosts **B** and **C**. The monitor (host **D**) will only see the communication between **A** and **C**, but not the one between **A** and **B**. Thus, in this example, the monitor will miss any ISN exchanged between **A** and **B**.

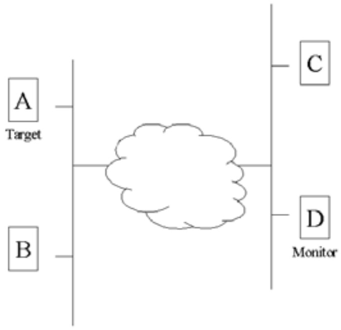


***Figure 1.*** *Limitation due to the position of the monitor when doing sequencing analysis*

The algorithms used to analyse how the ISNs are generated may also be troublesome. It is not always easy to determine whether or not numbers are randomly generated, nor is it obvious how to distinguish among random number generators.

The test PassiveTest_TCP_ISN is based on *nmap*'s TSeq test. To construct a sample, *nmap* initiates six consecutive connections with the target, and captures the SYN/ACK packets it receives in response. If at least 4 responses are received and that the delay between the probes[9] is no longer than one second, *nmap* considers the sample as being suitable for its calculations.

*nmap* classifies ISNs into several categories:

- Constant ISNs, i.e. OS always starts a connection using the same ISN (e.g. Commodore 64);
- ISNs that are multiple of 64000 (e.g. old UNIX and MAC OS prior to OS 9);
- ISNs that are multiple of 800 (e.g. IBM OS/2);
- ISNs incremented using random positive increments (most OSes);
- ISNs randomly generated (e.g. OpenBSD 2.9 or higher);
- Time dependent ISNs, i.e. ISN is incremented by a small fixed amount each time period (e.g. Windows 95/98/NT);

The algorithms *nmap* uses to classify the target into these different categories are based on computation of standard deviation and greatest common divisor,

[9] *nmap* has an option that allows the user to set the delay between the transmission of each stimulus. The longer the delay, the stealthier the tool is. However, in cases where the location of the system running *nmap* prevents it from seeing all of its target's traffic, a longer delay increases the likelihood of getting non-consecutive ISNs. This is because the target can communicate with other hosts during the sampling period.

made on the set of the differences between the Sequence Numbers (i.e. {SeqNb(i+1)-SeqNb(i)}, where SeqNb(i) is the Sequence Number of the $i^{th}$ packet sampled). The classification algorithms are simplistic. For example, if at least one of these differences is greater than 50 million, the sample is said to come from a true random generator. Similarly, *nmap* distinguishes between the class random incremental and the class time dependent based on the standard deviation value (if it is greater than a certain threshold, the sample falls into the random incremental class rather than the time dependent class without any consideration to the elapsed time). While definitely not infallible, *nmap*'s classification obtained for several tested targets seem to be consistent from sample to sample. For simplicity, we adopted the same algorithms.

The main difference between our test and *nmap*'s test is in the choice of packets composing the sample. In our case, we monitor both SYN and SYN/ACK packets sent by the target, no matter what the destination address is. *Nmap* captures only the SYN/ACK packets the target sends in response to its stimuli. Thus, for our test we make the assumption that the ISNs for SYN and SYN/ACK packets come from the same number generator, no matter what the OS is.

We impose a somewhat more restrictive condition on the sample: while the minimum number of packets is still four, all packets must arrive within one second of the first packet. Actually, the "four packets", and "one second" (1000ms) are default values of user-defined parameters in the prototype program. These default values were chosen based on observations made during the analysis of two traffic traces (section 5) containing web traffic. First, it appeared to be common for a host to initiate two to five TCP sessions within one second. This is because many hosts were using Netscape as their Browser, which uses separate TCP sessions to download a page containing several components (e.g. text, images). Secondly, these conditions are so restrictive that it is unlikely that other unseen TCP connections can be initiated in between, no matter where the monitor is located. Note that if the position of the monitor allows it to see all traffic of all hosts connected to the network, then the "within one second" restriction on the sample is futile. However, keeping a time restriction can prevent getting a sample with packets sent before and after a machine reboots.

The single field analysed in this test is found in the TCP header. The criteria defined are:

1. What class best describes the ISNs sampled? ("C"= Constant ISNs, "64K"= ISNs that are multiple of 64000, "i800"= ISNs that are multiple of 800, "RI"= ISNs incremented using random positive increments, "TR"= ISNs randomly generated, "TD"= Time dependent ISNs)

2. When the Class is "C", what is the value of the ISNs? (four-byte integer expressed in decimal)

3. When the Class is “TD” or “RI”, what is the greater common divisor of the ISN differences? (four-byte integer expressed in decimal)

4. When the Class is “TD” or “RI”, what is the value of the standard deviation of the ISN differences? (four-byte integer expressed in decimal)

Signature examples: ISNClass=TD;gcd=1;std=50;

Note that as in the case of *nmap*’s approach, the greater common divisor (gcd) and the standard deviation (std) are defined in the signature database in terms of lower and upper bound values. This is because the samples contain so few packets that these measures may vary a lot between samples. Thus, on each sample we compute the greater common divisor and the standard deviation, but when we look for a match in the database, we try to find one for which the lower and upper bounds are satisfied. We describe in section 4.2.2 how we estimated those upper and lower bounds.

The collected signatures appear in Table 14 of Annex A. The testing of these signatures on real user traffic indicates that the range delimited by the lower and upper bounds of the standard deviation is not wide enough to capture all possibilities. Note however that even if a perfect match is not found, the *ISNClass* field of the signature does help distinguishing between operating system families.

### 3.2.4 PassiveTest_IP_ID (Sample)

This test tries to categorize the target based on its IP ID number generation (see section 2.3.4). It is based on *nmap*’s TSeq test. A sample of six packets received within one second is inspected according to a modified version of *nmap*’s classification algorithms.

This test presents the same reliability problems described for the PassiveTest_TCP_ISN test due to the sampling process and analyzing algorithms.

While *nmap* only looks at the IP ID of SYN/ACK packets it receives from the target in response to its stimuli, we inspect all IP packets (not only those carrying TCP segments). We do this because most of the systems keep track of the IP ID value of the last IP datagram they have sent in order to produce the next value. Said differently, the IP IDs of two consecutive IP datagram sent by a host are very likely to be related to one another. Therefore, if we only capture the value for some special kinds of datagram, we would miss some sequenced IP IDs in between.

However, since we capture all IP traffic, we had to revise the *nmap* classification given below. For some Linux systems for instance, the IP ID is incremental within a TCP session, but starts at a random number each time a new TCP session begins.

The IP ID classes defined by *nmap* are:

- IDs incremented by one each time (Class I)
- IDs incremented by 256 each time (Class BI)[10]
- IDs incremented using random positive increments (Class RPI)
- IDs coming from a random distribution (Class RD)
- Repeatable IDs (Class C)
- Zeroed out IDs (Class Z)

The purpose of the class Z is to catch the few systems that zero out the IP ID in their SYN/ACK packets. Linux kernels 2.4.4 to 2.4.21(at least) fall into this category. In fact, Linux 2.4.xx routinely set the IPID field to 0x00 unless fragmentation is permitted. The packet types with IPID set to zero depend on the kernels. We observed that Linux 2.4.0-2.4.3 zero out the IPID for all packets having the DF bit set to 1 and the MF bit set to 0, while Linux 2.4.4-2.4.21 zero out the value for a subset of these packets, in particular, for ICMP Echo Requests (and ICMP Echo Replies in the Linux 2.4.4 case), UDP carrying DNS messages, TCP SYN/ACK, TCP RST/ACK, and some TCP ACK packets responding to FIN/ACK packets.

For Linux kernels 2.4.4 and above, when the IP ID is nonzero, it is session dependant. This means that several counters are running at the same time, one per (*source*, *destination*, protocol) triple[11]. Each counter is initialised randomly.

As described below, we have defined a subtest, of type *Singleton*, to capture the zeroing behaviour. Packets with IPID equalled to zero are filtered out of the samples passed to PassiveTest_IP_ID. We have removed the Class Z from our version of the test based on samples, and added four new categories:

[10] "BI" stands for "broken increment". This 256-incremental behaviour is seen on some little endian platforms when the operating system "forgets" to reorder the bytes. See section 2.3.4 for more details.

[11] When the protocol is TCP, the *source* and *destination* are defined by the pairs (source IP address, source port) and (destination IP address, destination port) respectively. However, when the protocol is UDP, the *source* and *destination* are simply the source IP address and the destination IP address (i.e. no matter what the ports are). Moreover, in the case of UDP, it appears that when the IP More Fragment bit is set to 0, the destination address does not influence the IP ID. This means these versions of Linux will have a global IP ID increment if they communicate using small UDP packets with different destination hosts. Finally, in the case of ICMP, the *source* and *destination* are defined by the source and destination address solely.

- IDs are session dependent, but incremented by one within a given session, (Class I-SD)

- IDs are incremented by one globally, i.e. session independent (Class I-SI).

- IDs are session dependent, but incremented by 256 within a given session,
(Class BI-SD)

- IDs are incremented by 256 globally, i.e. session independent (Class BI-SI).

The Class I is thereby a subset of both Classes I-SD and I-SI. When a sample contains packets showing an incremental behaviour (incremented by one) but all packets are within the same (*source*, *destination*, protocol) triple, then one cannot distinguish between I-SD and I-SI and the sample will simply fall into Class I. Same remark applies to Classes BI, BI-SD, and BI-SI.

The fields examined are found in the IP header. The criteria we define are:

1. What Class best describes the IDs sampled? (I, BI, RPI, RD, C, Z, I-SD, I-SI, BI-SD, BI-SI)

2. What protocol over IP do the packets carry? (The integer value of the IP Protocol field if all packets carry the same protocol, -1 otherwise)

Signature example: IPIDClass=I-SI;Protocol=-1;

The I-SD and BI-SD categories were defined based on Linux's behaviours. It is only once the prototype code was written that we realize that Mac OS prior to Mac OS X and Solaris systems also maintain different counters, but in these cases it is one counter per destination address (independently from the protocol or the port numbers). Because of the way the code is written, the results produced for these systems may appear contradictory. For example, a TCP packets sample will produce a signature I-SD if it contains communications with at least two different destination IP addresses, but may also produce I-SI if this sample contains communication with a single interlocutor, but involving different port numbers. At the time of writing the signature database contains entries with both I-SD and I-SI signatures for these Mac OS and Solaris systems.

To complement this test, we have added two subtests:
**PassiveTest_Echo_IP_ID**, and **PassiveTest_NULL_IP_ID**.
PassiveTest_Echo_IP_ID is of type *Stimulus-Response* and examines whether the IP ID of a Response is echoed from the IP ID of the stimulus.
PassiveTest_NULL_IP_ID is of type *Singleton* and looks for a null IP ID (i.e. IP ID with a zero value). These two subtests are useful in pinpointing certain

versions of OSes and are also useful for removal from the sampling process of PassiveTest_IP_ID the packets with an echoed or null IP ID.

The criteria we define for the **PassiveTest_NULL_IP_ID** are:

1. Is the IPID value equal to zero? (Y or N)

2. What protocol over IP does the packet carry? (The integer value of the IP Protocol field)

3. What "purpose" does the packet serve? (If the protocol over IP is ICMP, the "purpose" is described using the ICMP type and Code fields; if the protocol is TCP, the "purpose" is described by the TCP flags; if the protocol is UDP, we tag the purpose as "none", in other cases the purpose is described as "UNKNOWN").

Signature Example: NullIPID=Y;Protocol=1;PacketType=0:0;

The criteria we define for the **PassiveTest_Echo_IP_ID** are:

1. Is the IPID value in the response identical to the value in the stimulus? (Y or N)

2. What protocol over IP does the response carry? (The integer value of the IP Protocol field)

3. What "purpose" does the response serve? (If the protocol over IP is ICMP, the "purpose" is described using the ICMP type and Code fields; if the protocol is TCP, the "purpose" is described by the TCP flags; if the protocol is UDP, we tag the purpose as "none", in other cases the purpose is described as "UNKNOWN").

4. What protocol over IP does the stimulus carry? (The integer value of the IP Protocol field)

5. What "purpose" does the stimulus serve? (refer to criterion 3.).

Signature Example:
IPIDEcho=Y;Protocol=1;PacketType=0:0;StimulusProcolol=1;StimulusPacketType=8:0;

The signatures observed for IP ID settings of the different OSes appear in Table 15, Table 16 and Table 17 of Annex A. Table 15 contains the result of subtest PassiveTest_ECHOED_IP_ID. To reduce the size of the table, only the echoed cases have been reported in this table. Table 16 contains the results of subtest PassiveTest_NULL_IP_ID for Linux systems (other OSes do not zero out the IPID). There are Linux versions with two signatures for the same type of packet, one signature with NullIPID=Y and the other with NullIPID=N. Refer for instance to the signatures for Linux 2.0.x- 2.2.x. This is typical of Linux versions that have an incremental IP ID that starts at zero on reboot. Therefore, in most cases the IP ID is nonzero, and will be equal to zero only in the first packet transmitted after reboot. Linux 2.4.4-2.4.21 kernels have two signatures for a TCP ACK segment. These systems sometimes send a null IPID in a TCP ACK segment in response to a FIN/ACK packet. Table 17 contains the result of PassiveTest_IP_ID that analyze samples. The examination of the IP ID field is particularly useful for identifying Linux systems and distinguishing between the different kernel versions of this family. This is because the Linux implementation of the IP ID has distinctively changed from version to version. Moreover, certain Linux distribution can be recognized. In particular, it appears that S.u.S.E. distributions starting with S.u.S.E. 8.1 (kernel 2.4.19-4GB) do not zero out the IP ID.

### 3.2.5 PassiveTest_TCP_TS (Sample)

This test tries to categorize the Timestamp clock update rate and is also based on *nmap*'s TSeq test.

As before, the main difference between this test and *nmap*'s test is in the packets composing the sample. *Nmap's* samples are composed of SYN/ACK packets responding to its stimuli (SYN packets having the TCP timestamp option set). *Nmap*'s targets that do not support this option will fall into the “Timestamp Unsupported” category. In contrast, we capture all SYN and SYN/ACK packets having the TCP timestamp option set. Therefore, the samples we get come from targets that do support this option. The *nmap*'s “Timestamp Unsupported” class does not apply here and thus we do not define it. Note however that this information can be gained with another test of ours (PassiveTest_TCP_SYNACK, see section 3.2.8). Also note that a lot of systems do support this option but won't advertise it in their SYN packets. Thus, a sample containing only SYN/ACK packets may indicates that the target is from this category.

The Timestamp classes defined by *nmap* are:

- Timestamp clocks updated twice per second (Class 2HZ)
- Timestamp clocks updated 100 times per second (Class 100HZ)
- Timestamp clocks updated 1000 times per second (Class 1000HZ)
- Timestamp option unsupported by the OS (Class UNSUPPORTED)
- Timestamp option set but having a value of zero (Class Z)

We have added another category:

- Timestamp clocks updated 500 times per second (Class 500HZ)

This rate was observed for Linux kernels distributed by Red Hat version 8 (i.e. Red Hat modified kernels from 2.4.18-14 to at least 2.4.18-18.8.0).

While the Timestamp option may appear in any TCP packet, we chose to monitor only SYN and SYN/ACK packets in order to pinpointing certain OS versions that send a timestamp value of zero during the three-way handshake. For instance Windows 2000 systems wait until the connection is established before sending nonzero values. Thus, using our sampling method, a sample coming from a machine running windows 2000 will have values of zero in all its packets (Class Z). Moreover, Windows 2000 is from the category that does not advertise the option in their SYN packets, thus the sample will contain only SYN/ACK packets. NetBSD has adopted the same behaviour since the release 1.6. In contrast with Windows, these systems advertise the timestamp option in SYN packets as well.

In contrast with the tests PassiveTest_TCP_ISN and PassiveTest_IP_ID, the accuracy of the test is not impaired if the monitor misses packets during the sampling process. The restrictions on the sample can therefore be set more loosely. However, because the timestamp clock is in some cases related to the uptime of a machine [19], it can reset to zero when a system reboots. Having a delay restriction can prevent getting a sample with packets sent before and after a machine reboots.

The field under examination is found in the TCP header. The criterion we define is:

1. What class best describes the timestamp clock update rate? (Z, 2HZ, 100HZ, 500HZ, 1000HZ)

Signature example: TSClass=100HZ;

The signatures collected for this test appear in Table 18 of Annex A. Obviously, only the operating systems that support the TCP Timestamp option are present in this table.

The OSes that stand out the most based on this test are the Windows systems, recent releases of NetBSD, and Linux 2.4.18-14. The latter distinguishes itself from the other Linux versions by incrementing the timestamp clock five hundred times per seconds instead of one hundred times per second. This kernel comes with the Red Hat 8 distribution. Also tested but not included in the table was the kernel update 2.4.18-17 (also particular to Red Hat 8). This kernel also showed the same update rate as Linux 2.4.18-14.

Another observation made during the development of this test is that setting the timestamp value to zero in a SYN packet (as currently done by NetBSD) produces an undesirable effect if the packet is destined to a Linux system. While Linux does support the option and normally responds favourably to it, it will not collaborate in this particular case. This means that although the timestamp option is supported by both parties and was requested by the initiator of the connection, it will not be use at all during the communication.

Assuming the practice of zeroing out the timestamp value during connection set-up becomes so common that new operating systems are undistinguishable based on this test, it may become necessary to remove the Class Z from this test, and examine the update rate on *all* TCP packets with nonzero timestamp values instead of restricting the sampling to SYN and SYN/ACK packets. Note that a subtest of type *Singleton* could be defined to observe this zeroing behaviour in SYN and SYN/ACK packets. This subtest could be defined to listen for all TCP SYN and SYN/ACK packets and produce a signature based on the following criteria:

1. Is the TCP timestamp option set? (Y or N)

2. Is the timestamp value of this option set to zero? (Y or N)

3. What flags are set in the TCP header? (S or SA)

### 3.2.6 PassiveTest_ARP_Retransmit (Sample)

This test is based the proof-of-concept program called *Induce-ARP* described in section 2.1. It consists in observing the number of times an unanswered ARP Request is retransmitted, and by analyzing the delays separating the retransmissions. This situation of retransmitted ARP request happens for instance when a host tries to contact an unused (or temporarily unreachable) IP address on his subnet.

As with the PassiveTest_ARP_Request, this test can only be done on hosts sharing the same broadcasting environment as the monitor (ARP packets are not routed).

The sequence of unanswered ARP requests for the MAC address of a unique IP address is analyzed based on 4 criteria:

1. How many packets in total does the system send? (integer, this counts the first and the retransmitted packets)

2. What is the minimum delay between any two consecutive packets? (in microseconds)

3. What is the maximum delay between any two consecutive packets? (in microseconds)

4. What class best describes the delays between retransmission of packets? (“C”= Constant, “LI”= Linear Increment, “OI”= Other than linear Increment)

Signature example:
NbOfPackets=6;DelayMin=1000000;DelayMax=1000000;ARPClass=C;

Section 4.2.6 describes the signature collection process used to identify in which category each operating system falls. The signatures are provided in Table 19 of Annex A. While the network stack implementation of Linux systems resembles in many aspects to the implementation in BSD families, this test helps recognizing the Linux systems from the others. It also helps distinguishing between the different kernel versions of Linux. Mac OS (prior to version X) and Sun OS can also be distinguished from the other families easily.

A major difficulty with this test is to detect whether an unanswered ARP request is reissued because of the ARP module itself, or because a higher-level module made several requests. The following example with Windows systems illustrates the problem. The ARP module of Windows machines do not retransmit ARP request. Thus suppose a Windows machine sends one echo request (one ping) to an unreachable IP address on its subnet, then only one ARP request will be seen. But suppose now that the Windows machine sends multiple consecutive ping attempts to the unreachable IP address, then for each of these attempts the ARP module will send one ARP request. This later case may be falsely interpreted as being an ARP retransmission situation. Because our test does not currently attempt to identify what triggered the ARP module to send ARP requests, the program is likely to produce false results.

Note also that the test does not attempt to match ARP requests with ARP replies. This means that the case of an ARP request that receives an answer may be mistaken with the case of no retransmission. To compensate, the program only tries to find a match if the number of identical ARP requests seen is greater than one.

### 3.2.7 PassiveTest_ICMP_ID_SEQ (Sample)

This test was inspired by Arkin's observations regarding the ping utility of Windows and Unix systems [7]. The *ping* utility is included with most platforms to allow testing for TCP/IP connectivity using ICMP Echo messages. In most cases, *ping* is a command-line utility, although there are some Graphical User Interface (GUI) implementations.

Different *ping* implementations set the ICMP *Identifier* (ICMP ID) and ICMP *Sequence Number* (ICMP Seq) parameters differently as discussed in 2.3.7. PassiveTest_ICMP_ID_SEQ combines the two parameters together to differentiate between OS families and versions. To the best of our knowledge, there is currently no implementation of these techniques for passive OS identification.

To fingerprint a host, PassiveTest_ICMP_ID_SEQ requires collecting a sample of ICMP Echo Requests sent by that host. To be suitable for examination, the sample collected must include Echo Requests directed at a minimum of two different destination addresses. This is required to determine the behaviour of the ICMP ID field.

The ICMP ID and ICMP Sequence Number fields are analyzed based on 4 criteria:

1. What Class best describes how ICMP IDs are generated? ("C" for constant, "I" for incremental, "TDI" for Time Dependent Incremental (i.e. likely to be tied to process ID), "RD" for randomly generated)

2. What invariant characterize the ICMP ID values? (the ICMP ID value itself if class is “C”, the greatest common divider of the increments if class is incremental (class “I” or “TDI”), or –1 if no invariant is identified)

3. What Class best describes how ICMP Sequence Numbers are generated? ("C" for constant, "I" for incremental, "IGlobal" for globally incremental)

4. What invariant characterize the ICMP Sequence Numbers? (the value of the ICMP Sequence Number itself if class is “C”, the value of the increment if class is incremental (class “I” or “IGlobal”), or –1 if no invariant is identified)

Signature Example:
ICMPIDClass=C;IDInvariant=200;ICMPSeqClass=IGlobal;SeqInvariant=100;

Invariants are expressed in hexadecimal values. When describing the ICMP ID, the class “TDI” stands for Time-Dependent Incremental and indicate that the ICMP ID increases by increments of different size. For most OSes that fall into this category, the ICMP ID is tied to the Process ID (PID) associated with ping, and thus the increment between consecutive instances of *ping* will vary according to other processes in the system. Although the gaps are not directly related to the elapsed time between *ping* calls, the longer the lapse of time is, the more likely a number of processes in between will have been launched. The relation between the ICMP *Identifier* and the PID can be verified when the source code of *ping* is available. For FreeBSD systems for instance, this is found in the ping.c file located in /usr/src/sbin/ping/ and in which icmp_id takes its 16-bit value from getpid()&0xFFFF.

The signatures are provided in Table 27 of Annex A. Signatures for Mac OS 7 to 9 were produced with *MacTCP Ping* 2.0.2[12]. Because these older

[12] *MacTCP Ping* is an old, free, unsupported ping program from Apple Computer. *MacTCP Ping* 2.0.2 comes with a GUI and is available for download at

versions of Mac OS do not come with a ping utility by default, users are likely to install a different utility tool. In particular, a different behaviour was observed using another *ping* utility named Mac TCP Watcher 2.0[13]. With TCP Watcher, a new ICMP ID and a new ICMP Seq are generated each time the application is opened. The ICMP ID is fixed for each instance of the application (i.e. once it is opened), while the ICMP Seq is globally incremented (independent from the destination addresses).

The database does not currently include signatures for the Novell systems, except for Novell 4.11 that had an easily identifiable behaviour. While the ICMP Seq is easy to predict for Novell 5 and 6, the behaviour of the ICMP ID is not well understood. The *ping* utility of Novell NetWare systems come with a GUI from which users can ping multiple IP addresses simultaneously and on which *ping* statistics are displayed. The following describes observations made through experiments with the utility. When pinging several targets with the same instance of the *ping* application, the ICMP ID is the same for all targets, and the ICMP Seq is a global counter (increment=0x0100). This produces a signature with an ICMP ID of class “C”, and an ICMP Seq of class “IGlobal”. However, if the *ping* utility is closed and reopened, the ICMP Seq is reset to 0, and then is incremented (again by 0x0100), but the ICMP ID may change or not. If different, the new ICMP ID is equal to the previous plus and increment multiple of 0x0100. In this case, the signature produced has an ICMP ID of class “TDI” and an ICMP Seq of class “IGlobal”. If the computer is rebooted, the ICMP ID definitely changes, but still appears to be related to the previous value, leading again to a signature with an ICMP ID of class “TDI” and an ICMP Seq of class “IGlobal”. We chose not to include signatures for Novell 5 and 6 until the program is adapted to produce consistent signatures.

PassiveTest_ICMP_ID_SEQ performs well at distinguishing between families and between versions. Recall however, that in order to fingerprint a host, this test requires capturing a sample of ICMP Echo Requests directed at multiple destination IP addresses. It may therefore require monitoring traffic for a while in order to complete a proper sample. Because of this, we also define two subtests **PassiveTest_ICMP_ID** and **PassiveTest_ICMP_SEQ** to examine the ICMP *Identifier* and the ICMP *Sequence Number* independently. PassiveTest_ICMP_ID is of type Singleton and is performed on a packet of type Echo Request. It examines the value of the ICMP ID to determine if it is equal to one of the constant values used by Windows. It also examines the length and content contained in the ICMP data.

http://download.info.apple.com/Apple_Support_Area/Apple_Software_Updates/English-North_American/Macintosh/Misc/.

[13] *TCPWatcher* is a shareware application for testing TCP/IP networks. As with MacTCP *Ping*, *TCPWatcher* comes with a GUI. It is available for download from http://www.vicomsoft.com/ftp_site/help.ftp.html#Watcher

Fields under analysis for the **PassiveTest_ICMP_ID** are found in the IP and ICMP headers. The criteria we define are:

1. Is the ICMP ID value equal to a known constant value of Windows system? ("0x0100", "0x0200", "0x0300", or "other")

2. Is the IP *Don't Fragment* bit set? ("Y" or "N")

3. What *Type Of Service* is specified? (decimal value)

4. What is the length of the packet payload? (number of bytes)

5. What is the content of the packet payload? (in hexadecimal)

Signature Example:
ICMPID=other;DF=N;TOS=0;DataLen=12;ConstantData=0000;

Criterion 1 is meant to distinguish Microsoft systems from the other OS families. The data content signature (criterion 5) is restricted to the immutable portion of the ICMP data. In the Windows case, this data portion starts directly after the ICMP header; in the Unix-like case, it starts after the first 8 bytes, and in the Novell case it starts after the first 10 bytes. Unix-based *ping* utilities append the fixed data at an offset of 8-bytes after the end of the ICMP header; the first 8 bytes consisting of a timestamp used for the calculation of the round trip as described in the man page of ping. The Novell *ping* utility sends 12 bytes of data. The first 8 bytes appear to act as a timestamp (as with Unix), and the next two bytes behave like a counter per target. That is, for each new target this two-byte field starts at 0x0000 in the first Echo Request, it is then incremented by 0x0100 in the subsequent Echo Requests directed at that target[14]. The data portion that is fixed in the Novell case consists of the remaining two bytes.

PassiveTest_ICMP_SEQ is of type Sample; the sample contains ICMP Echo requests sent by a given host and directed at a single destination address. Since most *ping* utilities send by default several packets in order to collect statistics[15], such samples are likely to be seen. PassiveTest_ICMP_SEQ

[14] While this behaviour is exacly how the ICMP Sequence Number is used by Unix-like *ping* utilities, the ICMP Sequence Number is used differently by Novell's *ping*. More presisely, in the case of Novell, the two-byte field in the data portion is reset to zero when a new target is being pinged, while the ICMP Sequence Number field does not. This provides the Novell *ping* utility with two counters: one that is globally incremented independently from the target (giving the number of Echo Requests sent in total), and one that is specific for each target (giving the number of Echo Requests sent to that target).

[15] To test the reachability of the target, the *ping* utility of Windows system send four Echo Requests by default. The *ping* utility of Unix-based systems is configured by default to keep sending Echo Requests to the target until the user stops the application.

attempts to characterize the setting of the ICMP *Sequence Number* in a manner similar to the one performed by PassiveTest_ICMP_ID_SEQ. Since it analyses Echo Requests directed at a single target, this subtest cannot determine whether the *Sequence Number* act as a global counter or as a counter specific to each target. Therefore the class "IGlobal" defined for the test PassiveTest_ICMP_ID_SEQ does not apply.

The criteria we define for the **PassiveTest_ICMP_SEQ** are:

1. What Class best describes how ICMP Sequence Numbers are generated? ("C" for constant, "I" for Incremental behaviour)

2. What invariant characterize the ICMP Sequence Numbers? (the value of the ICMP Sequence Number itself if class is "C", the value of the increment if class is incremental (class "I), or –1 if no invariant is identified)

Signature Example: ICMPSeqClass=I;SeqInvariant=100;

The signatures for the two subtests are found in Table 28 and Table 29. As with PassiveTest_ICMP_ID_SEQ, the signatures for Mac OS versions 7 to 9 were produced using *MacTCP Ping* 2.0.2.

### 3.2.8 PassiveTest_TCP_SYNACK (Stimulus-Response)

This test is based on *nmap* test T1 (which sends a crafted SYN packet to an open port and analyses the SYN/ACK response).

PassiveTest_TCP_SYNACK is a *Stimulus-Response* type of test, where the stimulus is a SYN packet and the response is a SYN/ACK packet. In contrast with *nmap* T1 test, this is a passive test, thus it does not send any packet, and it only listens for matching pairs of SYN and SYN/ACK packets. It is the host sending the SYN/ACK packet that is being fingerprinted, but fields from both the SYN and the SYN/ACK packets are required for the analysis.

Fields under analysis are found in the IP and TCP headers. The response packets are analyzed based on eight criteria:

1. Is the IP Don't Fragment bit set in the in the response? ("Y" or "N")

2. What is the value of the IP Time To Live field in the response? (one-byte integer expressed in decimal format)

3. What is the value of the TCP Window size in the response? (decimal value, or n(MSS) if Window size is "n" times the Maximum Segment Size advertised in the SYN/ACK, or n(ReqMSS) if Window size is "n" times the Maximum Segment Size advertised in the SYN)

4. What is the TCP Acknowledgment number in the in the response (in relation to the sequence number of the triggering SYN packet)? (“S” if equal, “S++” if incremented by 1, “O” for any other value)

5. What TCP options are included in the in the response? (TCP options are ordered as they appeared in the packet, “M”= Maximum Segment Size, “N”= No-operation, “T”= Timestamp, “W”= Window Scale, “S”= SackOK, “L”= End Of List, “C.New”= Connection Counts New). We also capture the value for the Maximum Segment Size and Window Scale options using the format “@DecimalValue”.

6. What TCP ECN bits are set, if any, in the response? (“C”=CWR, “E”=Ecn-Echo)

7. Were the TCP ECN bits set in the stimulus (SYN packet)? (“C”=CWR, “E”=Ecn-Echo)

8. What is the set of TCP options requested by the stimulus (SYN packet)? (TCP options are sorted in alphabetic order and the NOP option is omitted from this set.)

Signature example:

DF=Y;TTL=64;WIN=12(MSS);AckNb=S++;TCPecn=;TCPopts=M@1460;SYN_ TCPecn=;SYN_SetofTCPopts={M@1460TW};

The signatures collected appear in Table 20 of Annex A. Criterion 3 examines the Window Size (WIN) value of the SYN/ACK. It can be observed from Table 20 that the WIN value of FreeBSD, Mac OS X, OpenBSD, Windows, SunOS 5.8 and 5.9, and NetBSD prior to 1.3 is a multiple of the Maximum Segment Size (MSS) advertised in the stimulus. It appears that the WIN value of SYN/ACK produced by certain OSes may also depend on the network service running, the window size advertised in the SYN and the TCP window scale option of the stimulus.

As described in section 4.2.7, we used several different SYN stimuli to collect the SYN/ACK responses from the operating systems attached to the testbed. The signature database would benefit from testing a wider range of network services and from including in the fingerprint the influence of the WIN value contained in the SYN.

During the signature collection process, we observed that the order in which the options are set in the SYN packet has no impact on the response. Therefore it is sufficient for criterion 8 to check only which options are set in the SYN packet, independently of the order in which they appeared, and of the padding in between options (the NOP). We believe the stimuli used to collect the signatures are representative (at least TCP option wise) of any SYN packet that can appear on the network in current implementations.

Taking the SYN into account generally helps to reduce the number of possible OSes. Take for instance the previous signature example. A simple database query reveals that this signature is associated with eleven FreeBSD versions (3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1, 4.2, 4.3). If we query the database for the number of signatures matching the first six criteria (i.e. DF=Y;TTL=64; WIN=12(MSS);AckNb=S++;TCPecn=;TCPopts=M@1460;), but not necessarily satisfying criteria 7 and 8, then the number of possible OS versions jumps to twenty-seven (FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, and also OpenBSD 2.9, 3.0, 3.1, 3.2, 3.3). This is generally the case for the other signatures produced by this test.

### 3.2.9 PassiveTest_TCP_RSTACK (Stimulus-Response)

PassiveTest_TCP_RSTACK is based on *nmap*'s T5 test (which sends a SYN packet to a closed port). It is similar to the "PassiveTest_TCP_SYNACK" test, but instead of looking for matching pairs of SYN and SYN/ACK packets, it looks for matching pairs of SYN and RST/ACK packets. It is the host responding with the RST/ACK packet that is being fingerprinted, but fields from both packets are required for the analysis.

SYN packets addressed to closed ports are normally rejected by mean of a RST/ACK packet. Note however that a RST/ACK response does not ensure that the targeted port is in the state CLOSED. The response may come from a filtering device responding on behalf of the target. If we have no information about the state of the targeted port, then the result of the test is less reliable.

Fields under analysis are found in the IP and TCP headers. The response packets are analyzed based on seven criteria:

1. Is the IP *Don't Fragment* bit set in the response? ("Y" or "N")

2. What is the value of the IP Time To Live field in the response? (one-byte integer expressed in decimal format)

3. What is the value of the TCP *Window size* in the response? (two-byte integer expressed in hexadecimal)

4. What is the TCP *Acknowledgment number* in the response (in relation to the sequence number of the stimulus)? ("S" if equal, "S++" if incremented by 1, "O" for any other value)

5. What TCP ECN bits, if any, are set in the response? ("C"=CWR, "E"=Ecn-Echo)

6. What TCP flags are included in the response? ("R"=RST or "RA"=RST/ACK) Note: If the flag is "R" alone, the *Acknowledgment*

*number* of criterion 4 is likely to be “O” since the field is invalid when the ACK flag is not set.

7. What TCP options are included in the response? (the string "echoed": if they are echoed from the stimulus, if not: a string containing the options ordered as they appeared). Note: Most systems don't send any TCP option when resetting a connection.

Signature Example:
DF=N;TTL=64;WIN=4000;AckNb=S++;TCPflags=AR;TCPopts=echoed;

The signatures collected appear in Table 21 of Annex A. Systems that are easily distinguished based on this test are the QNX systems because they echo the Window size and TCP options. SunOS 5.5 to 5.7 and Mac OS 7 to 8 also have a peculiar behaviour. They echo the TTL value of the stimulus in the response. Assuming they get probed by systems with different TTL values, they would be identified by PassiveTest_TCP_RSTACK since the results would converge to this subset of possible OSes.

### 3.2.10 PassiveTest_ICMP_Unreach (Stimulus-Response)

This test is based on *xprobe*’s and *nmap*’s tests that send out a UDP packet to a closed port to probe an ICMP port unreachable message. That is, the stimulus is a UDP packet sent to a closed port, and the Response is an ICMP port unreachable message (type=3, code=3).

Both of these active tools send the UDP packet to a port presumed to be closed. A difficulty in modifying the test so that it can be conducted passively is in defining a specific filter for the UDP traffic. Our approach is to capture all UDP packets, store them temporarily in memory[16], when an ICMP port unreachable message is seen, the program then searches for the corresponding stimulus in the allocated memory.

Fields under analysis are found in the IP, ICMP and UDP headers. The responses are categorized based on the following nine criteria:

1. Is the IP *Don’t Fragment* bit set in the response? (“echoed” if echoed from the stimulus, “Y” if the bit is set to one by default, or “N” if it is set to zero by default)

2. What is the IP *Time To Live* value in the response? (decimal value)

3. What Type of service (TOS) is set in the response? (“echoed” if echoed from the stimulus, or decimal value otherwise)

[16] A stimulus with no matching response is removed from the allocated memory once it gets two seconds old. The two-second is a default value for the user-defined parameter of a *Stimulus-Response* type of test.

4. How much bytes of the UDP Header and UDP data were returned in the response? ("all" if returned integrally, the number bytes returned otherwise)

5. Has the IP Total Length field of the offending packet been echoed correctly in the response? ("Y" if echoed correctly, "-" if less than the original value, or "+" if it is greater)

6. Has the IP *Identification* field of the offending packet been echoed correctly in the response? ("Y", or "N")

7. Have the IP *flags* and IP *dataOffset* fields been echoed correctly in the response? ("Y" if fields are echoed correctly, "0" if zeroed it out, "N" otherwise)

8. Has the IP checksum of the offending packet been zeroed out in the response? ("0", or "nonzero")

9. Has the UDP checksum of the offending packet been echoed correctly in the response? ("Y" if it is echoed correctly, "0" for zeroed out, "N" if incorrect, or "N/A" if not enough bytes of the datagram were returned to allow a check on this field)

Signature example:

DF=echoed;TTL=255;TOS=0;UDPLen=8;IntegIPLen=Y;IntegIPID=N;IntegIPFlags=N; IntegIPck=0;IntegUDPck=0;

Information on how these fields are used in fingerprinting can be found in the document describing *xprobe* [6].

While developing the signatures, targets were stimulated with different stimulus setting of the IP Don't Fragment bit and the IP Type Of Service to determine if these values were echoed back. More details regarding the signature collection process can be found in section 4.2.8. The signatures collected appear in Table 22 of Annex A. This test discriminates very well among the different OS families and among the different versions within those families. There are no signatures for BeOS and QNX 4.0 to 6.0. These systems did not respond to UDP packets destined to a close port.

### 3.2.11 PassiveTest_ICMP_Echo (Stimulus-Response)

This test is based on the *xprobe* test that sends an ICMP Echo request to its target. The stimulus is an ICMP Echo request (type=8) having a nonzero value for the ICMP code. The response packet is an ICMP Echo reply (type=8). As with all *Stimulus-Response* tests, it is the host sending the response packet that is being fingerprinted, but both the stimulus and the response are required for the analysis.

Fields under analysis are found in the IP and ICMP headers. The responses are categorized based on the following criteria:

1. Was there a response to the stimulus? (“Y” or “N”)

2. Is the IP *Don’t Fragment* bit set in the response? (“echoed” if echoed from the stimulus, “Y” if the bit is set to one by default, or “N” if it is set to zero by default)

3. What is the IP *Time To Live* value in the response? (decimal value)

4. What *Type Of Service* is specified in the response? (“echoed” if echoed from the stimulus, or decimal value otherwise)

5. What is the IP *Identification* value in the response? (“echoed” if equal to the IP ID of the stimulus, “0” if equal to zero, “nonzero” otherwise)

6. What is the ICMP code in the response? ("echoed" if the value is equal to the nonzero value of the stimulus, the decimal value otherwise)

Signature example: Resp=Y;DF=echoed;TTL=128;TOS=0;
IPID=nonzero;Code=0;

The signatures collected appear in Table 23 of Annex A. The reader may wonder why we check for responsiveness (first criterion) since most OSes are configured to respond to directed ICMP echo requests. We check for responsiveness because we noticed that some OSes, QNX 4.0/6.0 for instance, do not respond to ICMP echo Request when the ICMP code is nonzero. That is, they respond only if the ICMP code is 0. These OSes can therefore easily be identified by PassiveTest_ICMP_Echo if this test ever occurs.

Note that the stimulus we are looking for is in a sense abnormal because of the nonzero ICMP code. Therefore it should not be seen as part of normal traffic. Nonetheless, if such traffic appears, the test can identify Microsoft family systems. This is because the Microsoft family appears to be the only one to overwrite the *ICMP code* in their response. We decided to implement this test, even though the stimulus is unlikely to appear, in order to detect (and benefit from) this kind of “Microsoft give-away” probing. Nonetheless, a glance at the signatures produced leads us to believe that a modified version of this test, based on criteria 2, 3, 4, and 5, and performed on regular ICMP echo request / ICMP echo reply pairs would also be useful in distinguishing between families.

### 3.2.12 PassiveTest_ICMP_Info (Stimulus-Response)

This test is based on *xprobe* test that sends an ICMP Information request to detect whether or not a host responds to this kind of request. The stimuli are ICMP Information requests (type=15) and the responses are ICMP Information replies (type=16). The ICMP Information Request/Reply pair was intended to support self-configuring systems such as diskless workstations at boot time, to allow them to discover their IP address. Very few machines are configured to respond to information requests, especially since this mechanism is now obsolete as stated in RFC 1122[21]. There are presently other protocols a diskless machine can use to discover its own IP address. According to the author of *xprobe*, OpenVMS, HP UX 10.x, and SunOS 4.x do respond to ICMP Information requests. We do not have access to any of these systems to confirm.

Fields under analysis are found in the IP and ICMP headers. The responses are categorized based on the following criteria:

1. Was there a response to the stimulus? ("Y" or "N")

2. Is the IP *Don't Fragment* bit set in the response? ("Y" or "N")

3. What is the IP *Time To Live* value in the response? (decimal value)

4. What *Type Of Service* is specified in the response? (decimal value)

5. What is the IP *Identification* value in the response? ("echoed" if equal to the IP ID of the stimulus, "0" if equal to zero, "nonzero" otherwise)

Signature example: Resp=Y;DF=Y;TTL=255;TOS=0;IPID=nonzero;

All operating systems connected to the testbed silently discard any ICMP Information request (signature in this case is: Resp=N;DF=N/A;TTL= N/A;TOS= N/A;IPID= N/A; where "N/A" stands for Not Applicable). Please note that in the case of a non-response, it would be best to verify that the system is up and that the ICMP messages are not filtered before we conclude that this system is configured not to respond to ICMP Info requests. We have not addressed this issue for the prototype but we can think of a few passive or active techniques to discriminate between both cases. Passively, we could check for recent network activities, actively, we could inject an ARP request for targets that are on the same networks as the monitor. For other targets, we may consider sending an ICMP echo-request, or use stealth host discovery techniques such as those described in [22].

### 3.2.13 PassiveTest_ICMP_TS (Stimulus-Response)

This test is based on the *xprobe* test that sends an ICMP Timestamp request to detect whether or not a host responds to this kind of request. The stimuli are ICMP Timestamp requests (type=13) and the responses are ICMP Timestamp replies (type=14).

The ICMP Time Stamp request allows a node to query another for the current time. This allows the sender of the request to estimate the latency the network is experiencing. Typically, recent operating systems do implement this mechanism. For example, Windows 95/NT do not support it while Windows 98/Me/2000/XP do.

Fields under analysis are found in the IP and ICMP headers. The responses are categorized based on the following criteria:

1. Did the target respond to the stimulus? (“Y” or “N”)

2. Is the IP *Don’t Fragment* bit set in the response? (“echoed” if echoed from the stimulus, “Y” if the bit is set to one by default, or “N” if it is set to zero by default)

3. What is the IP *Time To Live* value in the response? (value in decimal format)

4. What *Type Of Service* is specified in the response? (“echoed” if echoed from the stimulus, or decimal value otherwise)

5. What is the IP *Identification* value in the response? (“echoed” if equal to the IP ID of the stimulus, “0” if equal to zero, “nonzero” otherwise)

Signature example: Resp=Y;DF=Y;TTL=255;TOS=echoed;IPID=0;

The signatures collected appear in Table 25 of Annex A. This test is useful in distinguishing between older and newer versions. Moreover, few OSes do not echo the DF bit and TOS field. Only the Windows family do not echo the TOS value; this family set the TOS to 0 independently of the requested TOS. OSes that do not echo the DF are NetBSD 1.6, 1.6.1, QNX 6.2.1, the Windows family, the Linux family, and the SunOS family. From the OSes that do not echo the DF bit, Linux 2.4.0-2.4.4 and SunOS set the DF to 1, and the others to 0. Moreover, while the TCP/IP stack implementation of SunOS resembles in many aspects to the implementation in Mac OS prior to X, this test helps discriminating between the two families since Mac OS systems prior to X do not respond to ICMP Timestamp Requests.

As mentioned earlier, in the case of a non-response, it would be preferable to verify that the system is up, that no filtering device blocks ICMP messages, and that the monitor has proper coverage before concluding that the system does not respond to ICMP Timestamp Requests.

### 3.2.14 PassiveTest_ICMP_Mask (Stimulus-Response)

This test is based on the *xprobe* test that sends an ICMP Mask address request to detect whether or not a host responds to this kind of request. The stimuli are ICMP Mask address requests (type=17) and the responses are ICMP Mask address reply (type=18). The ICMP Address Mask Request (and Reply) allows a node to determine what address mask is in use on a subnet to which it is connected. It was primarily intended for diskless workstation at boot time. RFC 1122 states that the implementation of the mechanism is entirely optional[17]. Recent operating systems tend not to respond to Mask Address requests. Therefore this test can discriminate between older and newer versions among a given family. Within the Windows family for example, Windows 95/98/NT (prior to NT service pack 4) respond to directed Mask address requests, while all more recent Windows systems do not. SUN Solaris systems (even newer ones) appear to be cooperative with this kind of query.

Fields under analysis are found in the IP and ICMP headers. The responses are categorized based on the following criteria:

1. Did the target respond to the stimulus? (“Y” or “N”)

2. Is the IP *Don’t Fragment* bit set in the target’s response? (“echoed” if echoed from the stimulus, “Y” if the bit is set to one by default, or “N” if it is set to zero by default)

3. What is the IP *Time To Live* value in the target’s response? (“echoed” if equal to the TTL of the stimulus, otherwise it is the value in decimal format)

4. What *Type Of Service* is specified in the target’s response? (“echoed” if echoed from the stimulus, or decimal value otherwise)

5. What is the IP *Identification* value in the target’s response? (“echoed” if equal to the IP ID of the stimulus, “0” if equal to zero, “nonzero” otherwise)

Signature example: Resp=Y;DF=Y;TTL=255;TOS=echoed;IPID=nonzero;

[17] This can be considered good practice, as the content of the reply would allow a malicious attacker to gain knowledge about a remote network’s configuration.

Signatures are found in Table 26. Aside from discriminating among OS versions based on responsiveness, this test is useful to identify certain OS versions because of peculiarities in the setting of the examined fields. For instance Mac OS 9.0 echoes the setting of the DF bit while all other OSes tested use a default setting, independent from the value in the stimulus. While most OSes echo the TOS value, the Windows family and Netware 5.1 set the DF bit to zero by default. Moreover, the Netware family can be easily identified because they echo the TTL value for this particular ICMP type.

Again, in the case of a non-response, it would be preferable to verify that the system is up, that no filtering device blocks ICMP message, and that the monitor has proper coverage before concluding that the system does not respond to ICMP Mask requests.

# 4. Collecting the signatures

This section describes how we collected the OS fingerprints for each test. The approach taken was to build our own database of fingerprints in a controlled private environment. Target operating systems were installed and queried methodically in the local testbed and the prototype was used to collect and store the signatures observed. While this is somewhat time consuming initially, it helped achieve control and uniformity over the testing process. The fingerprints can be found in Annex A.

## 4.1 Computer Network Testbed

The laboratory facility of the Network Security Research Group comprises a number of computers, switches, routers and network security devices to support research activities. It was decided that the testbed for this activity would include virtual machines in order to test a great variety of operating systems, without having to dedicate a large number of computers. Therefore, some computers in the testbed emulated several different guest operating systems. A guest operating system is basically encapsulated within a single file that acts as a virtual disk, thereby isolating it from the host system and other virtual machines.

Most of the virtual OSes in the testbed were installed under VMware [23]. VMware creates a virtualized x86 PC environment in which a guest operating system can run. VMware does not modify the behaviour of the host and guest operating systems. A host treats a guest workstation as an application, and as quoted from [23], “no modifications need to be made to the guest operating system when it is installed on a virtual machine.” Applications on a guest operating system run exactly as they do on a regular system. The virtual machines were configured to use bridged networking so that a guest operating system appeared as an additional computer on the same physical Ethernet network as the host. According to the vendor, one can run simultaneously as many guest OSes as the RAM allows for. Nonetheless, we refrained from simultaneously opening too many guest operating systems per host, to insure that neither the host nor its network adapter was overloaded.

VirtualPC is another software product we used to emulate few operating systems. This product provides low-level PC hardware emulation, allowing the installation of any PC-based operating system [24]. In particular, VirtualPC was used in the testbed to install older systems for which the installation in VMware was troublesome. The few host computers on which VMWare and VirtualPC were installed have Intel Pentium II (~450Mhz), III (~1Ghz), and IV(~1.8 Ghz) processors, a minimum of 512 MB of RAM, and disk drives capacities up to 120 GB.

The Macintosh virtual machines run on Apple hardware using Mac-on-Linux software [25]. The Apple computers were 600 MHz and 400 MHz PowerPC with a minimum of 256MB of RAM each.

With the exception of one SunOS version installed in VMware, the SunOS systems are not emulated and are installed on Sparc stations (UltraSparc 200MHz-300MHz and Axil311-320 (SS10 and SS20 clones)). The exception is with SunOS 5.8. This version was installed both on a UltraSparc workstation and on an Intel-based PC ( under VMware) to determine whether there were differences in the signatures of SunOS systems due to the processor technology (i.e. Intel versus Sparc).

Due to the use of virtual machines, the small testbed provided testers with access to close to 200 operating system versions from different families (Windows, Linux, OpenBSD, FreeBSD, NetBSD, SunOS, Macintosh, QNX, Novell, and BeOS). The list of systems installed is provided in Table 2 and Table 3. Linux kernels of Table 2 were downloaded from ftp://ftp.kernel.org/pub/, while Linux kernels in Table 3 were those packaged with different Linux distributions.

The testbed is depicted in Figure 2. It includes the target systems from which to collect the signatures (i.e. the systems to fingerprint), the passive OS detection prototype to construct these signatures, a monitor to record all network traffic in libpcap format (for further reference and re-learning purposes), and a few other computers used in certain cases to establish a communication with the systems under test. The signature collection process is discussed in the next section.

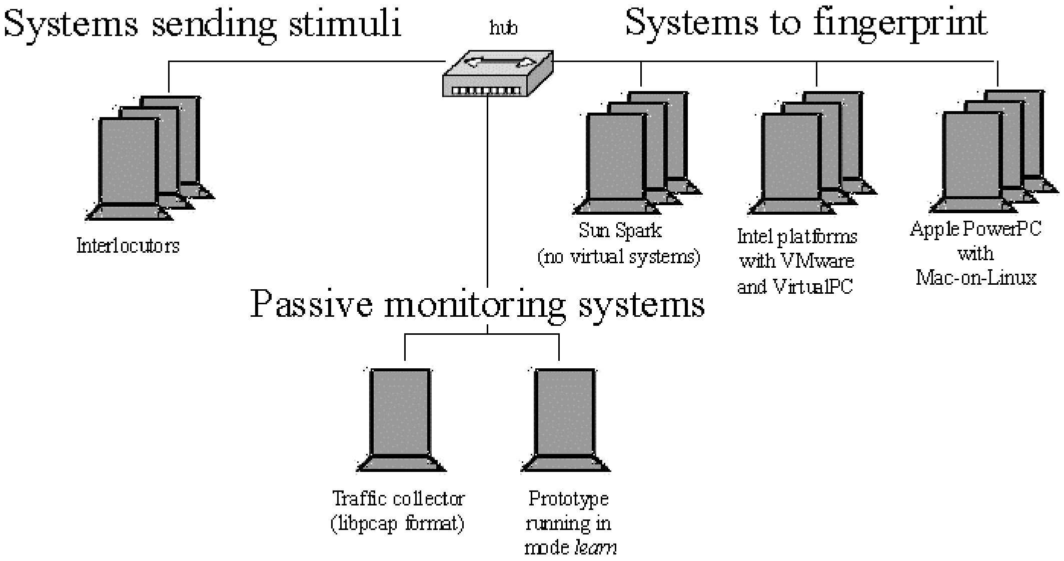

Systems sending stimuli
hub
Systems to fingerprint
Interlocutors
Sun Spark (no virtual systems)
Intel platforms with VMware and VirtualPC
Apple PowerPC with Mac-on-Linux
Passive monitoring systems
Traffic collector (libpcap format)
Prototype running in mode *learn*

***Figure 2.*** *Testbed from which the signatures were collected*

*Table 2. Tested Operating Systems*

| BeOS | FreeBSD | Linux 2.2.x (kernel.org) | Linux 2.4.x (kernel.org) | MacOS | NetBSD | Netware | OpenBSD | QNX RTP | SunOS | Windows |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 2.0.5 | 2.2.0 | 2.4.0 | 10.1.0 | 1.1 | 4.11 | 2.0 | 4 | 5.5 | NT 3.51 |
| | 2.1.0 | 2.2.2 | 2.4.1 | 10.1.1 | 1.2 | 4.11 sp9 | 2.1 | 6 | 5.5.1 | 95 |
| | 2.1.5 | 2.2.1 | 2.4.2 | 10.1.2 | 1.2.1 | 5 | 2.2 | 6.1 | 5.6 | 98 |
| | 2.1.6 | 2.2.3 | 2.4.3 | 10.1.3 | 1.3 | 5 sp6a | 2.3 | 6.2 | 5.7 | 98 SE |
| | 2.1.7.1 | 2.2.4 | 2.4.4 | 10.1.4 | 1.3.1 | 5.1 | 2.4 | 6.2.1 | 5.8 | Me |
| | 2.2.0 | 2.2.5 | 2.4.5 | 10.1.5 | 1.3.2 | 5.1 sp6 | 2.5 | | (Intel) 5.8 | 2000 |
| | 2.2.1 | 2.2.6 | 2.4.6 | 10.2.1 | 1.3.3 | 6 | 2.6 | | 5.9 | 2000 sp2 |
| | 2.2.2 | 2.2.7 | 2.4.7 | 10.2.2 | 1.4 | 6 sp3 | 2.7 | | | 2000 sp3 |
| | 2.2.5 | 2.2.8 | 2.4.8 | 10.2.3 | 1.4.1 | | 2.8 | | | 2000 sp4 |
| | 2.2.6 | 2.2.9 | 2.4.9 | 10.2.4 | 1.4.2 | | 2.9 | | | NT 4 |
| | 2.2.7 | 2.2.10 | 2.4.10 | 10.2.5 | 1.4.3 | | 3.0 | | | NT 4 sp3 |
| | 2.2.8 | 2.2.11 | 2.4.11 | 10.2.6 | 1.5 | | 3.1 | | | NT 4 sp4 |
| | 3.0 | 2.2.12 | 2.4.12 | 7.5.3 | 1.5.1 | | 3.2 | | | NT 4 sp6 |
| | 3.1 | 2.2.13 | 2.4.13 | 7.5.5 | 1.5.2 | | 3.3 | | | XP Home |
| | 3.2 | 2.2.14 | 2.4.14 | 7.6 | 1.5.3 | | | | | XP Pro |
| | 3.3 | 2.2.15 | 2.4.15 | 7.6.1 | 1.6 | | | | | Net |
| | 3.4 | 2.2.16 | 2.4.16 | 8.0 | 1.6.1 | | | | | 2003 |
| | 3.5.1 | 2.2.17 | 2.4.17 | 8.1 | | | | | | |
| | 4.0 | 2.2.18 | 2.4.18 | 9.0 | | | | | | |
| | 4.1 | 2.2.19 | 2.4.19 | 9.1 | | | | | | |
| | 4.1.1 | 2.2.20 | 2.4.20 | 9.2.1 | | | | | | |
| | 4.2 | 2.2.21 | | 9.2.2 | | | | | | |
| | 4.3 | 2.2.22 | | | | | | | | |
| | 4.4 | 2.2.23 | | | | | | | | |
| | 4.5 | 2.2.24 | | | | | | | | |
| | 4.6 | | | | | | | | | |
| | 4.6.2 | | | | | | | | | |
| | 4.7 | | | | | | | | | |
| | 4.8 | | | | | | | | | |
| | 5.0 | | | | | | | | | |
| | 5.1 | | | | | | | | | |

*Table 3. Tested Operating system: Linux distribution*

| Distribution | Debian | Redhat | S.u.S.E | Mandrake |
|---|---|---|---|---|
| **Releases** | 1.3 (kernel 2.0.29) | 4.2 (kernel 2.0.30) | 7.2 (kernel 2.4.4-4GB) | PPC 9.1 (kernel 2.4.21-0.13mdk) |
| **(and kernels)** | 2.0 (kernel 2.0.34) | 5.0 and 5.1 (kernel 2.0.32) | 7.3 (kernel  2.4.10-4GB) | |
| | 2.1 (kernel 2.0.36) | 5.2 (kernel 2.0.36) | 8.0 (kernel 2.4.18-4GB) | |
| | 3.0 (kernel 2.2.20-idepci) | 6.0 (kernel 2.2.5-15) | 8.1 (kernel 2.4.19-4GB) | |
| | | 6.1 (kernel 2.2.12-20) | | |
| | | 6.2 (kernel 2.2.14-5) | | |
| | | 7.0 (kernel 2.2.16-22) | | |
| | | 7.1 (kernel 2.4.2-2) | | |
| | | 7.3 (kernel 2.4.18-3) | | |
| | | 8.0 (kernel 2.4.18-14) | | |
| | | 9.0 (kernel 2.4.20-8) | | |

## 4.2 Stimulation Procedures and Traffic Capture

Analysis of the code implementation in open source systems was often used as a starting point for constructing the signatures, but not for determining the signatures. Signatures contained in the database were produced from network traffic collected from the testbed. Target operating systems were installed and queried methodically in the local testbed and the prototype was used to collect and store the signatures observed. While this was somewhat time consuming initially, it helped achieve control and uniformity in the testing process. Signatures for each test were collected separately.

The general procedure was the same for all tests:

1. Stimulate each target sequentially (one after the other), either from a remote machine or directly from the target depending on the test;

2. Capture the traffic with a monitor located on the same network segment as the targets, and save the traffic trace in the *tcpdump* (libpcap) format. This traffic trace file was accompanied by a text document describing how the targets were stimulated and by a list of these targets (i.e. their IP addresses (which were private and static) with their operating systems).

3. Once the traffic trace was complete, it was analysed by the prototype program running in the *learning* mode.

When running in *learning* mode, the program interfaces with a database containing several tables. First, there is a table that associates each IP address with its operating system. This table is managed manually and updated each time a new system is installed. Then, for each test there are two tables: one containing the distinct signatures, and one that associates an operating system with one or sometimes several[18] signatures. Some tests, the PassiveTest_TCP_RSTACK for instance, have few distinct signatures (few entries in their signature tables). This is because several different systems respond identically to such tests. Keeping the signatures apart from their OS associations helps in recognizing tests that best distinguish between operating systems. Thus in the *learning* mode, the program analyses the traffic, seeking suitable packets to be tested. When packets are found, the prototype generates the fingerprint, adds the fingerprint to the appropriate signature table, and creates the association between this signature and the operating system[19] in the second table.

When associating an IP address to an operating system in the database, the operating system description is broken down into several fields to allow comparison between outputs. The fields that make up the descriptions are “Type”, “Release”, “Version”, and “Kernel”. “Type” refers to the family (e.g. Windows, Mac OS, etc.). “Release”

[18] For example, this is the case for the PassiveTest_TCP_SYNACK, for which each OS has been stimulated during the signature collection process by eight different stimuli, each requesting a particular set of TCP options.

[19] The program can obtain the OS associated with an IP address by querying the table that contains this information for hosts connected to the computer network testbed.

may be empty depending on the family. In the Windows case for instance, this field corresponds to NT, XP, 2000, etc. and in the Mac OS case it corresponds to 7, 8, 9, and 10. The “Version” provides further details such as the service pack for Windows or a specific Mac OS 10 (Mac OS X) version such as 10.2.2. The “Kernel” field is used for Linux systems.

The procedures adopted when stimulating the targets are briefly discussed in the following sections. Note that when capturing traffic traces, we required that no other activities took place on the network testbed.

### 4.2.1 PassiveTest_TCP_SYN

This test requires a single packet, namely a TCP SYN segment, from the target. To force the transmission of a SYN packet, we used the web browser, the telnet and ftp clients on each operating system, initiating hereby communications with a server connected to the testbed. The signatures are shown in Table 12 of Annex A.

### 4.2.2 PassiveTest_TCP_ISN

This test analyses Sequence Numbers found in the SYN and SYN/ACK packets. It requires much attention when generating signatures. We generated 60 samples (of 6 packets each) for every operating system. Several samples were required in order to express the greatest common divisor (gcd) and the standard deviation (std) of a sample in terms of lower and upper bound values in the signature table. This lower and upper bound approach was adopted because a sample contains so few packets that these measures may vary a lot from one sample to another. When the program runs in a mode different than *learn*, it computes the gcd and std on a sample, and tries to find a signature in the database for which the two computed values fall into the two respective ranges. The ranges for the gcd and std are thus estimated based on the 60 samples captured. The signatures are shown in Table 14 of Annex A.

To capture these samples, we used a third party tool called *hping2* [26]. Using this tool from a remote host connected to the testbed, a series of crafted SYN packets were sent to each target (360 SYN packets in total per target). Each SYN packet was aimed at an unfiltered open port. For each stimulus the target responded with a SYN/ACK segment announcing the initial Sequence Number for the referenced connection. Upon reception of the SYN/ACK, *hping2* terminated the connection by sending a RST.

We observed that some targets do not handle the RST properly, and retransmit the SYN/ACK. Linux machines with kernel 2.4.18 and above, fall into this category. The observed behaviour of these Linux systems is the following: they respond to the SYN with 6 SYN/ACK segments (i.e. 5 retransmissions). They wait 3 seconds before sending the first retransmission, then 6, 12, 24, and finally 48 seconds between the last two. This mishandling of the RST packet during the connection set-up was observed for few other systems. This situation is not unique to the packet generator used; it holds for any other tool that can perform a so-called *half-open SYN scan* on its target.

While one can make an active OS detection test based on these observations, this retransmission behaviour causes problems for the prototype program that analyses the traffic. The program does not currently detect such duplicates. This causes samples containing old retransmitted ISNs to be processed. When we observed a target having this problematic behaviour, we removed all traffic regarding this OS from the tcpdump trace, and stimulated this target otherwise, using a complete three-way handshake connection set-up. To do this we used a shell script calling *nmap* with the –sT option (in a “for” loop).

### 4.2.3 PassiveTest_IP_ID

This test, of type Sample, captures all IP traffic. Based on the observation that some OSes maintain several IP ID counters running at the same time, one per (source, destination, protocol) triple, we tried to generate traffic that would mix ICMP, TCP, and UDP communications, and as far as possible, we tried to mix it based on the following parameters: Protocols, IP destination addresses, source and destination ports. The goal was to generate enough samples so that the program could determine whether the IP ID generation is session dependent or not. The approach taken was to utilize two shell scripts: one installed on the machine under test (which is often platform dependent), and one install on two different interlocutors. The system under test initiates different communications with both interlocutors, which in turn also initiate communications with the target. The signatures are shown in Table 17 of Annex A.

### 4.2.4 PassiveTest_TCP_TS

This test requires capturing TCP SYN or SYN/ACK packets having the timestamp option set. To generate the signatures for this test, we used the same traffic traces collected for the PassiveTest_TCP_ISN. Note that the crafted stimuli (SYN packets) had the TCP option turned on. The signatures are shown in Table 18 of Annex A.

### 4.2.5 PassiveTest_ICMP_ID_SEQ

This test requires capturing ICMP Echo Requests transmitted by the *ping* utility installed by default on the system. There is no ping utility installed by default on Mac OS versions 7 to 9. As mentioned in section 3.2.7, a free tool from Apple Computer was installed for testing. For most OSes, the *ping* utility consists of a command line program. A shell script was written to execute *ping* commands in sequence. The script was conceived to allow the capture of six samples containing six Echo Requests each. Different samples test have different test purposes. In particular, we were interested in examining the impact of pinging different targets and having other processes running. The same traffic traces were used to collect the signatures for the subtests PassiveTest_ICMP_ID and PassiveTest_ICMP_SEQ. The signatures are found in Table 27, Table 28 and Table 29.

### 4.2.6 PassiveTest_ARP_Request and PassiveTest_ARP_Retransmit

Both of these tests capture ARP Request packets. PassiveTest_ARP_Request requires a single packet, while PassiveTest_ARP_Retransmit requires a sample of such packets. To generate the traffic, the targets were stimulated using induce-ARP (described in section 2.1). The signatures are shown in Table 13 and Table 19 of Annex A.

### 4.2.7 PassiveTest_TCP_SYNACK and PassiveTest_TCP_RSTACK

To get the SYN/ACK and RST/ACK packets, targets were probed from eight different machines. These machines were selected for their different sets of TCP options they used to initiate a connection. The different sets of TCP options of the stimuli are summarised in the following table.

***Table 4. Stimuli used to collect SYN/ACK and RST/ACK signatures***

| OS used for sending the stimulus | Set of TCP options contained in the stimulus | TCP options ordered as they appeared in the stimulus |
|---|---|---|
| Windows NT4 | {M} | M@1460 |
| QNX 6.0 | {M@1459} | M@1459 |
| Windows 2000 | {M,S} | M@1460NNS |
| OpenBSD 2.6 | {M,T,W} | M@1460NW@0NNT |
| NetBSD 1.6 | {M,T@0,W} | M@1460NW@0NNT@0 |
| FreeBSD 2.2.8 | {C.New,M,T,W} | M@1460NW@0NNTNNC.New |
| Linux 2.4.7 | {M,S,T,W} | M@1460STNW@0 |
| OpenBSD 2.9 | {M,S,T,W} | M@1460NNSNW@0NNT |

The TCP options of the Windows NT 4 and QNX 6.0 stimuli differ by the value of the MSS option. The QNX 6.0 stimulus was chosen to detect OSes that echo the MSS value in their answer. The OpenBSD 2.6 and the NetBSD 1.6 initiate a connection with the same set of TCP options, except that NetBSD 1.6 advertised a TSval of 0. The effect of this practice is that some OSes that normally support the option will not participate in any timestamp exchange. More explicitly, these machines will show support for the Timestamp option in their response to OpenBSD 2.6, but not in their response to NetBSD 1.6. This is the case for Linux 2.4.0 that responds with TCPopts= M@1460**NNT**NW@0 when responding to a stimulus with SYN_SetOfTCPopts={M@1460TW}, but with TCPopts= M@1460NW@0 when responding to a stimulus with SYN_SetOfTCPopts= {M@1460T@0W} (see signatures for Linux 2.4.xx from Table 20). The FreeBSD 2.2.8 stimulus was chosen because it advertises a non-standard option (experimental status) as described in section 3.2.1. The last two OSes (Linux 2.4.7 and OpenBSD 2.9) have the same TCP options in their SYN but placed in a different order. We wanted to verify by this means that the order in which options are set in the SYN packet does influence the order in which the supported options are set in the response. Except for Mac OS 7.5.3 to 8.1[20], all other OS tested responded identically whether the SYN came from OpenBSD 2.9 or from Linux 2.4.7. Thus when constructing the signatures, it seems sufficient to check only which options were set in the SYN packets, independently of the order in which they appear, and of the padding in between options (the NOP). This practice helps reducing the number of stimuli required to express all possible SYN packets.

The choice of these 8 stimuli to collect the signatures is representative (at least TCP option wise) of the SYN packets that can be transmitted by any of the OS systems attached to the testbed.

From each of the computers in Table 4, we used the *telnet* client six times. We specified each time a different destination port number in the *telnet* command: three known to be open, and three known to be closed. The signatures are shown in Table 20 and Table 21 of Annex A for PassiveTest_TCP_SYNACK and PassiveTest_TCP_RSTACK respectively.

### 4.2.8 PassiveTest_ICMP_Unreach/Echo/Info/TS/Mask

For these five Stimulus-Response tests, we stimulated the targets using *hping2* and an in-house modified version of *xprobe1*-0.0.2. *Xprobe1*-0.0.2 functions according to a logic tree and thus terminates as it reaches a leaf. Depending on the branch of the tree, the stimulus it sends may differ. The in-house modified version of *xprobe* produces a signature-based output based on the responses to all of *xprobe*'s packets: a UDP packet aimed at a closed port, an ICMP Echo request with a nonzero ICMP code value, an ICMP Timestamp request, an ICMP Mask Address request, and a ICMP Information request. In addition, a shell script calling *hping* was used to craft ICMP requests with different TOS and DF values to determine whether or not the responses would echo the values contained in the stimuli. The signatures are shown in Table 22 to Table 26 of Annex A.

[20] Experiments conducted on the testbed lead us to believe that these systems do not process the last TCP option appearing in the SYN packet. Aside from the NOP and the EOL options, these machines only support the MSS and Window scale option. They show support for both options when stimulated by OpenBSD 2.9, but the Window Scale option is missing from their response to Linux 2.4.7.

# 5. Field Test Evaluation

This section presents preliminary results obtained from two traffic traces captured in November 2002 using a tapping device located on the intranet web server’ s segment of the corporate network. Both traffic traces were captured using *tcpdump* with a filter specified to monitor IP traffic from and to 59 IP addresses. The tapping device, a Shomiti UTP Tap IL/1 from Finisar Company, was installed in cooperation with the Corporate Network Systems section to tap the internal web server’s line. The content of the two traffic traces is summarised in Table 5 and Table 6. Because of the tap’s location, the traffic captured consists mainly of HTTP traffic. There would also have been some ARP traffic if the capture filter had not filtered it out. This omission is somewhat unfortunate since these traffic traces do not allow the testing of the ARP based tests. Note that while the capture filter was specified to capture traffic involving 59 hosts[21], the total number of different IP addresses seen is 50 in Trace 1, and 55 in Trace 2. The total number of IP addresses includes the count of hosts being monitored and hosts with whom they have communicated.

***Table 5. Traffic Trace #1***

File format: libpcap
Capture filter: (defined to capture traffic to/from 59 hosts)
Dates: November 13 to 15, 2002
Elapsed time: 2 days, 5 hours, 10 minutes, and 40.614 seconds
Packet count: 8714
Avg. packets/sec: 0.046
Bytes of traffic: 3815796
Avg. bytes/sec: 19.932 (159.456bps)
IP addresses seen in total: 50

| **Protocols** | **Bytes** | **%Bytes** | **Packets** | **%Packets** |
|---|---|---|---|---|
| **Ethernet** | **3815796** | **100** | **8714** | **100** |
| **Internet protocol** | **3815796** | **100** | **8714** | **100** |
| **Internet Control Message protocol (ICMP)** | **120** | **0.003145** | **2** | **0.022952** |
| **Transmission Control Protocol (TCP)** | **3815676** | **99.99686** | **8712** | **99.97705** |
| Hypertext transfer protocol (HTTP) | 3807068 | 99.771267 | 8619 | 98.9098 |
| Non-HTTP traffic | 8608 | 0.2255886 | 93 | 1.0672481 |

[21] The operating system and version information for each of the monitored hosts was also determined manually for comparison purposes.

***Table 6. Traffic Trace #2***

File format: libpcap
Capture filter: (defined to capture traffic to/from 59 hosts)
Dates: November 25 to 29, 2002
Elapsed time: 4 days, 3 hours, 40 minutes, and 21.397 seconds
Packet count: 14779
Avg. packets/sec: 0.041
Bytes of traffic: 7451145
Avg. bytes/sec: 20.766 (166.128bps)
IP addresses seen in total: 55

| **Protocols** | **Bytes** | **%Bytes** | **Packets** | **%Packets** |
|---|---|---|---|---|
| **Frame** | **7451145** | **100** | **14779** | **100** |
| **Ethernet** | **7451145** | **100** | **14779** | **100** |
| **Internet protocol** | **7451145** | **100** | **14779** | **100** |
| **Internet Control Message protocol (ICMP)** | **3662** | **0.049147** | **32** | **0.216523** |
| **Transmission Control Protocol (TCP)** | **7447483** | **99.95085** | **14747** | **99.78348** |
| Hypertext transfer protocol | 7423894 | 99.634271 | 14447 | 97.753569 |
| Non-HTTP traffic | 23589 | 0.3165822 | 240 | 1.6239258 |

Aside from the *learning* mode described in section 4.2, there are two other modes in which the prototype can run: *verify* and *find*. When run in either of these modes, the program interfaces with the database in which the signatures and corresponding OS associations are stored. When a test produces a signature that cannot be found in the database, a mechanism to look for an alternative signature is called upon. The current mechanism tries to find a match considering solely the most important fields of a given test. The "importance" of each field of a test is indicated with a binary weight. A weight of 1 indicates that the value must match exactly; while a 0 means that the field is not required to match. Consider for instance the test PassiveTest_TCP_SYN which consists in 5 fields: DF, TTL, WIN, TCP_Ecn; and TCPopts. We have assigned a value of 0 for two of these fields: WIN and TCP_Ecn. This means that if this test produces a signature for which no perfect match[22] is found, the program will search the database for alternative signatures matching the fields DF, TTL, TCPopts. To illustrate, suppose a TCP SYN packet is captured and produces the following signature:

DF=Y;TTL=128;WIN=faf0;TCP_Ecn=;TCPopts=M@1460NNS.

This signature does not appear in the database as it can be seen from Table 12, but two alternative signatures can be found with different Window Size values:

- DF=Y;TTL=128;WIN=2000;TCP_Ecn=;TCPopts=M@1460NNS,
  Associated with Windows 98 and 98 SE;

- DF=Y;TTL=128;WIN=4000;TCP_Ecn=;TCPopts=M@1460NNS,
  Associated with Windows Me, Windows 2000 standard, sp3, sp4, sp6, Windows XP Home, Professional, Windows Net and Windows 2003.

[22] All fields computed match with those of a certain signature contained in the database.

In this example, the program would informed the user that it had to look for alternative signatures, and the set of possible OSes found consists of Windows 98, 98 SE, Windows Me, Windows 2000 standard, sp3, sp4, sp6, Windows XP Home, Professional, Windows Net and Windows 2003.  Table 7 summarise the results of each test obtained with the mode verify.  It can be seen that the two traces allowed the testing of the following tests only:

- PassiveTest_TCP_SYN,
- PassiveTest_TCP_SYNACK,
- PassiveTest_TCP_TS,
- PassiveTest_TCP_ISN,
- PassiveTest_IP_ID,
- PassiveTest_ICMP_ID_SEQ,
- PassiveTest_ICMP_ID, and
- PassiveTest_ICMP_SEQ.

The columns entitled “# of good results” give the number of times a test produced a set of possible OSes that does not conflict with the actual OS of the systems being tested. The columns “# of false results” counts the number of times a test produced a mismatch between the true OS and the set of possible OSes found.  Finally, the columns “# of results that cannot be verified” indicates the number of instances of a test that were performed on IP addresses for which the true OS is unknown, and thus for which the outcome cannot be verified.  When it applies, the table indicates the number of times a signature was obtained through the alternative signature mechanism.  For example, for the test PassiveTest_TCP_SYN in Trace 1, 57 out of 523 signatures did not find a perfect match, but found a match using the alternative signature mechanism.  Some of the signatures obtained from Trace 2 were new signatures (i.e. never seen on the testbed).  In such cases the outcomes were empty sets because no match (or alternative match) was found in the database.

*Table 7. Results of individual tests*

| | Trace #1 | | | Trace #2 | | |
|---|---|---|---|---|---|---|
| | **# of good results** | **# of false results** | **# of results that cannot be verified** | **# of good results** | **# of false results** | **# of results that cannot be verified** |
| **ARP Request** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ARP Retransmit** | 0 | 0 | 0 | 0 | 0 | 0 |
| **TCP SYN** | 523<br>(incl. 57 alt) | 0 | 0 | 678<br>(incl. 151 alt) | 0 | 0 |
| **TCP SYN/ACK** | 522 | 0 | 0 | 626 | 0 | 0 |
| **TCP RST/ACK** | 0 | 0 | 0 | 0 | 0 | 0 |
| **TCP Timestamp** | 9 | 0 | 0 | 15 | 2 | 0 |
| **TCP ISN** | 22 | 2 | 0 | 37<br>(incl. 1 alt) | 1 | 0 |
| **IP ID** | 1417 | 121 | 0 | 2507 | 206 | 2 |
| **ICMP Echo** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ICMP Info** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ICMP Mask** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ICMP Timestamp** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ICMP Unreachable** | 0 | 0 | 0 | 0 | 0 | 0 |
| **ICMP ID SEQ** | 0 | 0 | 0 | 0 | 0 | 1 (a new signature) |
| **ICMP ID** | 0 | 0 | 2 | 4 | 0 | 26 (3 new signatures) |
| **ICMP SEQ** | 0 | 0 | 0 | 1 | 0 | 0 |

When analysing Trace #2, PassiveTest_ICMP_ID produced three new signatures for which we could not verify the OS. Those signatures are:

1. ICMPID=other ;DF=N ;TOS=0 ;DataLen=24 ;ConstantData=UNKNOWN ;
2. ICMPID=other ;DF=Y ;TOS=0 ;DataLen=470 ;ConstantData=UNKNOWN ;
3. ICMPID=100 ;DF=N ;TOS=0 ;DataLen=9 ;ConstantData=UNKNOWN ;

For the host with the third unknown signature, PassiveTest_ICMP_ID_SEQ, which examines how the ICMP Identifier and the ICMP Sequence Number vary, also found a new signature:

1. ICMPIDClass=C ;IDInvariant=100 ;ICMPSeqClass=I ;SeqInvariant=100 ;

Some of the tests require parameter settings as described in section 3.1; these settings appear in the first two columns of Table 8. Timeouts are in milliseconds (column 1) and apply to tests of types *Stimulus-Response* and *Sample*. A timeout value of “-1” indicates that the prototype can wait indefinitely (until the end of the traffic trace) to complete the sample. The third column indicates which fields were given a weight value of 0 for the alternative matching signature mechanism. These settings were determined empirically and may require changes for different environments.

As mentioned in section 3.2 describing each test, some tests contain fields for which the values depend on the network environment. These fields are the IP Time to Live (TTL) and the TCP option Maximum Segment Size (MSS). The signatures in the database contain TTL and MSS values for local hosts sharing an Ethernet link. The look-up algorithm for these fields works in a fashion similar to the alternative signature mechanism. If no exact match is found, it searches the database for plausible values. The details of the matching algorithms for the TTL and MSS are very intuitive and are thus not presented here.

***Table 8. Tests Parameters***

| | **Parameters** | | **Fields with Weight 0 if alternative signature mechanism is required** |
|---|---|---|---|
| | **Timeout (ms)** | **sample size (# of packets)** | |
| **ARP Request** | | | |
| **ARP Retransmit** | 1500 | | |
| **TCP SYN** | | | WIN, TCPecn |
| **TCP SYN/ACK** | 2000 | | WIN, ackNb, TCPecn, SYN_TCPecn, |
| **TCP RST/ACK** | 2000 | | ackNb, TCPecn, SYN_TCPecn, |
| **TCP ISN** | 1000 | 4 | val, gcd, std |
| **IP ID** | 1000 | 6 | |
| **TCP Timestamp** | -1 | 6 | |
| **ICMP Echo** | 2000 | | |
| **ICMP Info** | 2000 | | |
| **ICMP Mask** | 2000 | | |
| **ICMP Timestamp** | 2000 | | |
| **ICMP Unreachable** | 2000 | | |
| **ICMP ID Seq** | -1 | 18 | |

Each instance of a test is performed separately and its outcome (a set of possible Operating Systems) is passed to an intersection module. This module manages the information coming from all the tests and attempts to identify the set of possible OSes on which all tests agree. The only difference between the mode *verify* and *find* is that the former checks whether the true operating system associated with an IP address is known, and if so, it verifies if the outcome of the test produces a mismatch.

Table 9 and Table 10 give the results for each individual host obtained with the prototype running in *verify* mode. The IP addresses in the tables do not correspond to the real IP addresses. They have all been anonymized to preserve the privacy of end users and of the campus's allocated IP address range. An IP address that appears in both traces will appear as the same anonymized IP address across those traces. The IP addresses tested belong to hosts being monitored and hosts with whom they have communicated. The number of IP addresses for which communication was sufficient to conduct at least one test is 29 in both traces. Of these 29 IP addresses, there were 28 IP addresses for which the true OS was known, either exactly (24/28) or approximately (4/28). There also was one IP address in each trace with an unknown OS.

*Table 9. Results obtained from Traffic Trace #1 for each host*

| Traffic Trace #1 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.1.55 | Linux 2.4.2-2 | TCP_SYN, IP_ID, TCP_ISN, TCP_TS | OS MISMATCH due IP_ID test [23] | Other tests (including other instances of IPID tests) agreed on 26 results:<br>[Linux 2.4.2-2, Linux 2.4.4, Linux 2.4.4-4GB, Linux 2.4.5, Linux 2.4.7, Linux 2.4.6, Linux 2.4.8, Linux 2.4.9, Linux 2.4.10, Linux 2.4.10-4GB, Linux 2.4.11, Linux 2.4.12, Linux 2.4.13, Linux 2.4.14, Linux 2.4.15, Linux 2.4.16, Linux 2.4.17, Linux 2.4.18, Linux 2.4.18-3, Linux 2.4.18-4GB, Linux 2.4.18-14, Linux 2.4.19, Linux 2.4.19-4GB, Linux 2.4.20, Linux 2.4.20-8, Linux 2.4.21-0.13mdk] |
| 192.168.5.175 | Linux 2.4.18-17 [24] | TCP_SYN, IP_ID, TCP_ISN, TCP_TS | 1 | [Linux 2.4.18-14] |
| 192.168.1.94 | MacOS 10 | TCP_SYN, IP_ID, TCP_TS | 1 | [MacOS 10.0.0] |
| 192.168.5.67 | MacOS 10 | TCP_SYN, IP_ID | 1 | [MacOS 10.0.0] |
| 192.168.1.69 | SunOS 5.7 | TCP_SYN, IP_ID | 4 | [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |
| 192.168.5.224 | SunOS 5.7 | TCP_SYN, IP_ID | 4 | [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |

[23] One IP_ID sample produced a [IPIDClass=RD, Protocol=6] signature while the true IPIDClass for protocol 6 (TCP) is either "IPIDClass=I" if all packets belong to the same TCP session or "PIDClass=I-SD" otherwise.

[24] This particular kernel does not appear in the signature database. However, Linux 2.4.18-17 is a kernel update for Redhad 9.0, which has kernel 2.4.18-14 in the CD installation. The database only contains the signature for 2.4.18-14.

| Traffic Trace #1 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.5.57 | SunOS 5.7 | TCP_SYN, IP_ID, TCP_ISN | OS MISMATCH due to TCP_ISN[25] | other tests agreed on 4 results [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |
| 192.168.1.106 | SunOS 5.8 | TCP_SYN, IP_ID | 1[26] | [SunOS 5.8] |
| 192.168.1.97 | SunOS 5.8 | TCP_SYN, IP_ID, TCP_ISN | 1[27] | [SunOS 5.8] |
| 192.168.1.15 | Windows 2000 | TCP_SYN, IP_ID, TCP_ISN | 3 | [Windows 2000 standard, Windows 2000 Server standard, Windows 2000 sp4] |
| 192.168.1.159 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.163 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.191 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.148 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.162 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |

[25] There was one ISN sample for this host. This sample produced the signature [ISNClass=RI,val=-1,gcd=2,std=6411]. The measured standard deviation (6411) is a little lower than what we obtained from all SunOS samples in the testbed. The alternative signature mechanism was not called upon since the signature found a match for a few OSes (a few old versions of FreeBSD and OpenBSD, along with Windows 2000 and Windows Me).

[26] TCP_SYN on its own was sufficient to find OS.

[27] TCP_SYN on its own was sufficient to find OS.

| Traffic Trace #1 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.5.12 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.105 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.74 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.219 | Windows 2000 sp2 | TCP_SYN, IP_ID, TCP_ISN | OS MISMATCH due to TCP_ISN[28] | other tests agreed on 9 results<br>[Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.3.16 | Windows 2000 sp2 | TCP_SYN, IP_ID, TCP_ISN | 4 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3] |
| 192.168.5.26 | Windows 2000 sp3 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.79 | Windows Millennium | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.188 | Windows NT 4 | TCP_SYN, IP_ID | 4 | [Windows NT 4 standard, Windows NT 4 sp3, Windows NT 4 sp4, Windows NT 4 sp6] |

[28] There was one ISN sample for this host in the trace. This sample produced the signature [ISNClass=RI,val=-1,gcd=1,std=27607]. The measured standard deviation (27607) is a little higher than what we obtained from the testbed for Windows 2000 machines. The alternative signature mechanism was not called upon since the signature had found a match in the database for several OSes, including a number of versions from FreeBSD, OpenBSD, MacOS and SunOS families. The only Windows system matching the signature was Windows Millenium.

| Traffic Trace #1 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.1.80 | Windows XP Professional | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.206 | Windows XP Professional | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.207 | Windows XP Professional | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.6.92 | Linux RH 6.2, kernel unknown [29] | TCP_SYNACK, TCP_ISN, TCP_TS, IP_ID | 3 | [Linux 2.2.14-5, Linux 2.2.16, Linux 2.2.20] |
| 192.168.3.154 | Windows | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| outsider | unknown | ICMP_ID | 7 | [Netware 4.11, Netware 4.11 sp9, Netware 5, Netware 5 sp6a, Netware 5.1, Netware 5.1 sp6, Netware 6 sp3] |

***Table 10. Results obtained from Traffic Trace #2 for each host***

| Traffic Trace #2 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |

[29] The only information we had about this host was the Linux distribution (i.e. Red Hat 6.2). The kernel version we tested within this Linux distribution was 2.2.14-5 (see Table 3), which was the basis kernel at installation of Red Hat 6.2.

| Traffic Trace #2 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.5.21 | Linux 2.4.18-14 | TCP_SYN, IP_ID, TCP_TS, ICMP_ID, ICMP_SEQ | 1[30] | [Linux 2.4.18-14] |
| 192.168.1.55 | Linux 2.4.2-2 | TCP_SYN, IP_ID, TCP_ISN, TCP_TS | OS MISMATCH due to IP_ID[31] | other tests (including other instances of the test IP_ID) agreed on 11 results:<br>[Linux 2.4.0, Linux 2.4.2-2, Linux 2.4.4-4GB, Linux 2.4.8, Linux 2.4.12, Linux 2.4.17, Linux 2.4.18, Linux 2.4.18-4GB, Linux 2.4.19, Linux 2.4.19-4GB, Linux 2.4.10-4GB] |
| 192.168.1.94 | MacOS 10 | TCP_SYN, IP_ID, TCP_TS | 1 | [MacOS 10.0.0] |
| 192.168.1.69 | SunOS 5.7 | TCP_SYN, IP_ID | 4 | [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |
| 192.168.5.224 | SunOS 5.7 | TCP_SYN, IP_ID | 4 | [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |
| 192.168.1.50 | SunOS 5.7 | TCP_SYN, IP_ID | 4 | [SunOS 5.5, SunOS 5.5.1, SunOS 5.6, SunOS 5.7] |
| 192.168.1.106 | SunOS 5.8 | TCP_SYN, IP_ID | 1 | [SunOS 5.8] |
| 192.168.1.97 | SunOS 5.8 | TCP_SYN, IP_ID, TCP_ISN | 11 | [FreeBSD 4.2, MacOS 9.2.1,NetBSD 1.3.1, NetBSD 1.3.2, NetBSD 1.3.3, NetBSD 1.3, OpenBSD 2.8, QNX RTP 6.1, SunOS (Intel) 5.8, SunOS 5.8, SunOS 5.9] |
| 192.168.1.150 | Windows 2000 | TCP_SYN, IP_ID, TCP_ISN | 3 | [Windows 2000 sp2, Windows Millennium standard, Windows 2000 standard] |
| 192.168.1.158 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |

[30] TCP_TS was sufficient to pin point the OS.

[31] One IP_ID sample produced a [IPIDClass=RD, Protocol=6] signature while the true IPIDClass for protocol 6 (TCP) is either “IPIDClass=I” if all packets belong to the same TCP session or “PIDClass=I-SD” otherwise.

| Traffic Trace #2 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.1.159 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.163 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.191 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.148 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.162 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.12 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.105 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.74 | Windows 2000 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |

| Traffic Trace #2 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.1.219 | Windows 2000 sp2 | TCP_SYN, IP_ID, TCP_ISN | OS MISMATCH due to TCP_ISN[32] | other tests agreed on the following results:<br>[Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.3.16 | Windows 2000 sp2 | TCP_SYN, IP_ID, TCP_ISN | 4 | [Windows 2000 sp2, Windows Millennium standard, Windows 2000 sp3, Windows 2000 Server standard] |
| 192.168.1.189 | Windows 2000 sp3 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.26 | Windows 2000 sp3 | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.5.188 | Windows NT 4.0 | TCP_SYN, IP_ID | 4 | [Windows NT 4 standard, Windows NT 4 sp3, Windows NT 4 sp4, Windows NT 4 sp6] |
| 192.168.1.80 | Windows XP Professional | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.206 | Windows XP Professional | TCP_SYN, IP_ID, ICMP_ID | 8 | [Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.1.207 | Windows XP Professional | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |

[32] There was one ISN sample for this host in the trace. This sample produced the signature [ISNClass=RI,val=-1,gcd=1,std=27079]. The measured standard deviation (27079) is a little higher than what we obtained from the testbed for Windows 2000 machines. The alternative signature mechanism was not called upon since the signature had found a match in the database for several OSes. The only Windows system matching this signature was Windows Millenium.

| Traffic Trace #2 | | | | |
|---|---|---|---|---|
| **IP addresses (sanitized)** | **Real OS** | **Test conducted** | **Number of OSes in set of OS match** | **OSes contained in set of OS match** |
| 192.168.6.93 | Linux RH 6.2 (kernel unknown) [33] | TCP_SYNACK, IP_ID, TCP_ISN, TCP_TS | 9 | [Linux 2.2.7, Linux 2.2.12-20, Linux 2.2.13, Linux 2.2.16, Linux 2.2.20, Linux 2.2.24, Linux 2.4.2-2, Linux 2.4.17, Linux 2.4.18] |
| 192.168.3.154 | Windows | TCP_SYN, IP_ID | 9 | [Windows Millennium standard, Windows 2000 standard, Windows 2000 sp2, Windows 2000 sp3, Windows 2000 sp4, Windows XP Home, Windows XP Professional, Windows Net standard, Windows 2003 Server standard] |
| 192.168.15.15 | unknown | IP_ID | 123 | the set of possible OSes **EXCLUDES** Linux 2.4.2-2, Linux 2.4.4-Linux 2.4.21, Novell Netware, OpenBSD 2.5-3.3, and Windows 95/98/NT |

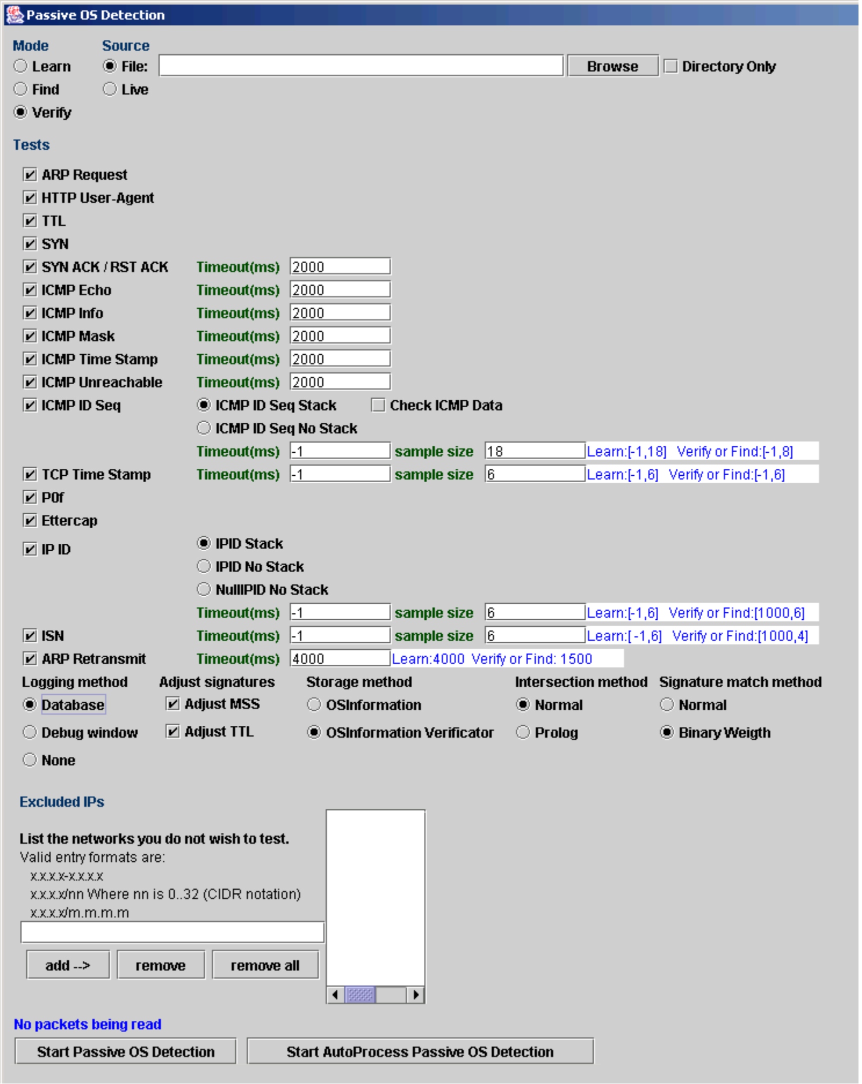
[33] The only information we had about this host was the Linux distribution (i.e. Red Hat 6.2). The kernel version we tested within this Linux distribution was 2.2.14-5 (see Table 3), which was the basis kernel at installation of Red Hat 6.2.

# 6. Development State of the Prototype

The passive OS detection prototype is one of a set of tools developed at CRC for network monitoring and analysis.  Although the prototype works as a standalone application, it was also integrated into two in-house information gathering tools: a Network Mapping Tool [27] that actively scans the network to discover topology and information available about the network components and a Passive Network Monitoring Tools [28] that gathers similar information without sending any packets.

The standalone version comes with three modes of operation: *learn*, *find*, *verify*.  All modes can run on live traffic or pre-recorded traffic traces.  The purpose of the *learn* mode is to collect the signature from an environment in which the association between OS and IP addresses is known.  The traffic traces used to learn the signature have been archived.  Therefore if a test is modified, the signatures for all OSes tested can be updated according to the change made.  Moreover, as new OS versions are being released, procedures are in place to install and test the new systems in order to update the database.

The *find* mode is the mode used on an environment for which we have little to no knowledge.  It allows acquiring OS information about the computers connected to the network.  The *verify* mode was developed for testing purposes.  The only difference between the *verify* mode and *find* mode is that the former checks whether the true operating system associated with an IP address is known, and if so, it verifies if the outcome of the test produces a mismatch.  This mode is typically used on live traffic or on traffic traces different than those from which the signatures were learned.  As displayed in Figure 3, the standalone version includes a Graphical User Interface (GUI) to select the traffic trace or network interface to monitor and to select the tests to activate.  Parameters for *Stimulus-Response* and *Sample* tests (see section 3.1) can be changed from that GUI.  The results are not displayed graphically and are sent to a database instead.



Passive OS Detection

Mode
○ Learn
○ Find
● Verify

Source
● File: ______ Browse ☐ Directory Only
○ Live

Tests

☑ ARP Request
☑ HTTP User-Agent
☑ TTL
☑ SYN
☑ SYN ACK / RST ACK Timeout(ms) 2000
☑ ICMP Echo Timeout(ms) 2000
☑ ICMP Info Timeout(ms) 2000
☑ ICMP Mask Timeout(ms) 2000
☑ ICMP Time Stamp Timeout(ms) 2000
☑ ICMP Unreachable Timeout(ms) 2000
☑ ICMP ID Seq ● ICMP ID Seq Stack ☐ Check ICMP Data
○ ICMP ID Seq No Stack
Timeout(ms) -1 sample size 18 Learn:[-1,18] Verify or Find:[-1,8]
☑ TCP Time Stamp Timeout(ms) -1 sample size 6 Learn:[-1,6] Verify or Find:[-1,6]
☑ P0f
☑ Ettercap
☑ IP ID ● IPID Stack
○ IPID No Stack
○ NullIPID No Stack
Timeout(ms) -1 sample size 6 Learn:[-1,6] Verify or Find:[1000,6]
☑ ISN Timeout(ms) -1 sample size 6 Learn:[ -1,6] Verify or Find:[1000,4]
☑ ARP Retransmit Timeout(ms) 4000 Learn:4000 Verify or Find: 1500

Logging method
● Database
○ Debug window
○ None

Adjust signatures
☑ Adjust MSS
☑ Adjust TTL

Storage method
○ OSInformation
● OSInformation Verificator

Intersection method
● Normal
○ Prolog

Signature match method
○ Normal
● Binary Weigth

Excluded IPs

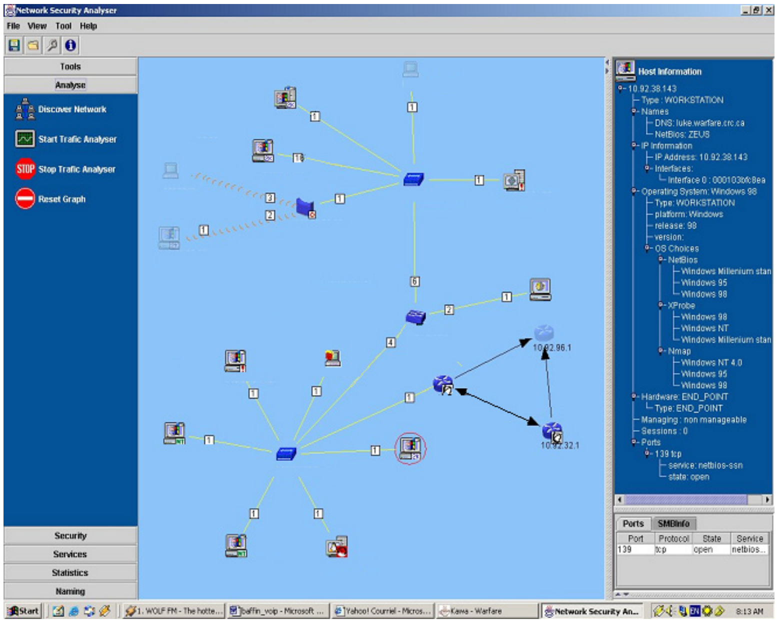
List the networks you do not wish to test.
Valid entry formats are:
x.x.x.x-x.x.x.x
x.x.x.x/nn Where nn is 0..32 (CIDR notation)
x.x.x.x/m.m.m.m

No packets being read

Start Passive OS Detection
Start AutoProcess Passive OS Detection

***Figure 3.*** *Test selection menu*

The two in-house Network Monitoring Tools provide the OS information and much more via a graphical user interface. When called from these tools, the OS identification tool runs in *find* mode. Figure 4 shows the connectivity and the description of diverse network components discovered using the active Network Mapping tool. In the left panel, computers and network devices are represented using intuitive icons. The icon indicates the OS family identified. The right panel displays more detailed information concerning the selected system (a Windows system in this case). The GUI included in the Passive Network Monitoring Tool is displayed in Figure 5. It provides an example of what an analyst would see once the passive monitoring tool has been running for a period of time. The screenshot shows several computers, a router, and a switch. The right panel displays more detailed information concerning the selected system (a Novell system in this case). The prototype is designed to display information in a summarized manner for users who are primarily interested in aggregated information, but also allows analysts to examine in more detail the performance of individual techniques.  Techniques used in these tools include the capability to discover active nodes, operating systems, the node's role in the network, system uptime, the services offered, the protocols supported, IP network interface configuration and the network topology at different level of specification (physical, logical).  The focus is on developing reliable techniques and mitigating the shortcomings found in available tools.  More information on the capabilities of the Network Mapping Tool and the Passive Network Monitoring tool can be found in [27] and [28].



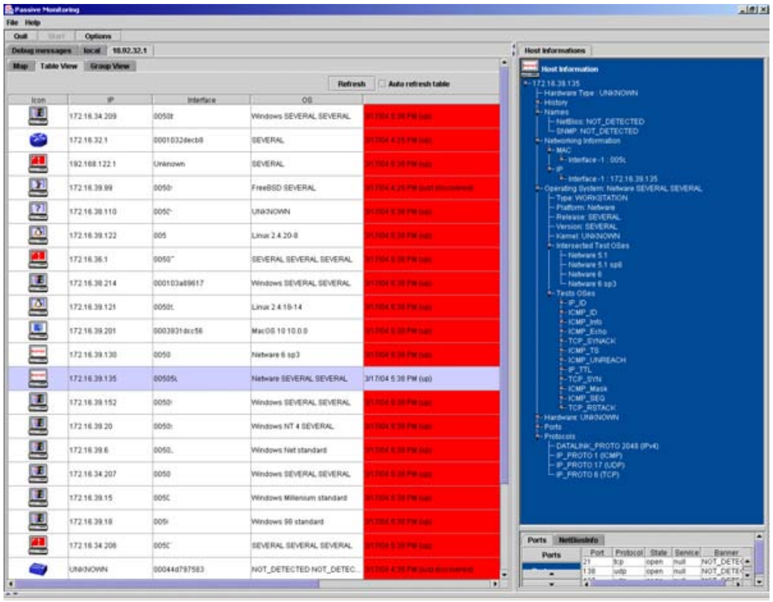
Host Information

```
10.92.38.143
├ Type : WORKSTATION
├ Names
│ ├ DNS: luke.warfare.crc.ca
│ └ NetBios: ZEUS
├ IP Information
│ ├ IP Address: 10.92.38.143
│ └ Interfaces:
│   └ Interface 0 : 000103bfc8ea
├ Operating System: Windows 98
│ ├ Type: WORKSTATION
│ ├ platform: Windows
│ ├ release: 98
│ ├ version:
│ └ OS Choices
│   ├ NetBios
│   │ ├ Windows Millenium stan
│   │ ├ Windows 95
│   │ └ Windows 98
│   ├ XProbe
│   │ ├ Windows 98
│   │ ├ Windows NT
│   │ └ Windows Millenium stan
│   └ Nmap
│     ├ Windows NT 4.0
│     ├ Windows 95
│     └ Windows 98
├ Hardware: END_POINT
│ └ Type: END_POINT
├ Managing : non manageable
├ Sessions : 0
└ Ports
  └ 139 tcp
    ├ service: netbios-ssn
    └ state: open
```

Ports | SMBInfo

| Port | Protocol | State | Service |
|---|---|---|---|
| 139 | tcp | open | netbios... |

***Figure 4.*** *CRC' s Active Network Mapping Tool GUI*



| Icon | IP | Interface | OS | |
|---|---|---|---|---|
| | 172.16.34.209 | 0050t | Windows SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.32.1 | 0001032decb8 | SEVERAL | 3/17/04 4:25 PM (up) |
| | 192.168.122.1 | Unknown | SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.99 | 0050 | FreeBSD SEVERAL | 3/17/04 4:25 PM (just discovered) |
| | 172.16.38.110 | 0050 | UNKNOWN | 3/17/04 5:38 PM (up) |
| | 172.16.39.122 | 005 | Linux 2.4.20-8 | 3/17/04 5:38 PM (up) |
| | 172.16.36.1 | 0050 | SEVERAL SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.38.214 | 000103a89617 | Windows SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.121 | 0050t | Linux 2.4.18-14 | 3/17/04 5:38 PM (up) |
| | 172.16.39.201 | 0003931dcc56 | MacOS 10.10.0.0 | 3/17/04 5:38 PM (up) |
| | 172.16.39.130 | 0050 | Netware 6 sp3 | 3/17/04 5:38 PM (up) |
| | 172.16.39.135 | 00505L | Netware SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.152 | 0050 | Windows SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.20 | 0050 | Windows NT 4 SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.6 | 0050 | Windows Net standard | 3/17/04 5:38 PM (up) |
| | 172.16.34.207 | 0050 | Windows SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | 172.16.39.15 | 005C | Windows Millenium standard | 3/17/04 5:38 PM (up) |
| | 172.16.39.18 | 005 | Windows 98 standard | 3/17/04 5:38 PM (up) |
| | 172.16.34.206 | 005C | SEVERAL SEVERAL SEVERAL | 3/17/04 5:38 PM (up) |
| | UNKNOWN | 00044d797583 | NOT_DETECTED NOT_DETEC... | 3/17/04 4:35 PM (just discovered) |

```
172.16.39.135
  Hardware Type : UNKNOWN
  History
  Names
    NetBios: NOT_DETECTED
    SNMP: NOT_DETECTED
  Networking Information
    MAC
      Interface -1 : 005L
    IP
      Interface -1 : 172.16.39.135
  Operating System: Netware SEVERAL SEVERAL
    Type: WORKSTATION
    Platform: Netware
    Release: SEVERAL
    Version: SEVERAL
    Kernel: UNKNOWN
    Intersected Test OSes
      Netware 5.1
      Netware 5.1 sp6
      Netware 6
      Netware 6 sp3
    Tests OSes
      IP_ID
      ICMP_ID
      ICMP_Info
      ICMP_Echo
      TCP_SYNACK
      ICMP_TS
      ICMP_UNREACH
      IP_TTL
      TCP_SYN
      ICMP_Mask
      ICMP_SEQ
      TCP_RSTACK
  Hardware: UNKNOWN
  Ports
  Protocols
    DATALINK_PROTO 2048 (IPv4)
    IP_PROTO 1 (ICMP)
    IP_PROTO 17 (UDP)
    IP_PROTO 6 (TCP)
```

| Port | Protocol | State | Service | Banner |
|---|---|---|---|---|
| 21 | tcp | open | null | NOT_DETE( |
| 138 | udp | open | null | NOT_DETE( |

***Figure 5.*** *CRC' s Passive Network Monitoring Tool GUI*

The structure of the code allows additional tests to be included easily. A new passive test named PassiveTest_IP_TTL has been developed to identify OSes that send different TTL values depending on the type of packet to transmit. PassiveTest_IP_TTL is of type Singleton and monitors all IP traffic. While most OSes allow the default TTL value to be changed, some OSes override this value for certain types of packets. For example Linux systems use a value of 64 in a datagram carrying a TCP segment, unless the RESET TCP flag is turned on, in which case, the TTL value is 255. PassiveTest_IP_TTL may identify computers for which the overridable default value has been changed.

Recent work within the team has focused on developing a Scenario-Driven Intrusion Detection System (SDIDS) based on temporal logic [29]. The SDIDS is an evolution of the Passive Network Monitoring Tool. This system has the ability to identify attack scenarios involving multiple packets and to passively gather information about the monitored network, providing thereby context with intrusion alarms. Traditional Intrusion Detection Systems tend to suffer from a high false-positive rate. Bringing context to intrusion detection can help address the false positive problem.

# 7. Limitations and Future Work

## 7.1 Limitations

### 7.1.1 Fingerprinting countermeasures

Whether they are passive or active, fingerprinting methods can be defeated. Filtering devices can be configured to reject or silently discard certain types of packets. Certain fields of packets transmitted may also be changed without breaking communications. These changes can take place at the endpoint transmitting the packets or at devices along the path. A few deception tools were developed to specifically defeat known OS identification tools. *IP Personality* is an example of a popular endpoint solution for Linux systems to defeat *nmap* by impersonating a different OS [30]. Certain firewalls and intrusion protection systems advertise having fingerprinting countermeasure functionality (e.g. Sygate's Personal Firewall [31]). However, this appears to be centred on the more usual practice of dropping abnormal packets such as those used by *nmap*. Therefore such solutions are less effective against passive fingerprinting techniques based on normal traffic.

While making custom changes to the TCP/IP stack is easier to accomplish on open source platforms, minor modifications (such as changing default values for configurable variable) can be done on virtually any OS. When the stack imitates the behaviours of an existing platform, the deception method is more likely to go undetected. This is because a plausible signature may be found in the database and thus lead to the wrong conclusion. In contrast, deception attempts based on producing unusual signatures are likely to draw attention. Experienced analysts may "reverse-engineer" the process or simply pay closer attention to other traffic in order to deduce the original OS.

M. Smart et al. proposed the use of a protocol scrubber to defeat TCP/IP stack fingerprinting [32]. The intended use of the fingerprint scrubber is to transparently interpose between the Internet and the network under protection. The goal of the tool is not to prevent fingerprinting when done internally, but rather to prevent OS information leakage to the exterior world. The OpenBSD packet filter is another example of a solution that allows end-nodes or gateways to do traffic normalization with scrub rules [33].

Simple mechanisms have been implemented in the prototype to allow individual test to look for alternative signatures when no perfect match is found. This typically counters simplistic attempts to defeat fingerprinting. In practice, we found that it is in the variety of the tests that OS information can leak through despite the attempts to defeat OS detection. Because the approach relies on several individual tests conducted of different protocols, it takes much effort to defeat all tests. One has to introduce changes in enough places to make the deception plausible.

A higher-level module in the prototype combines the outcomes of the tests to obtain a set of the most plausible OS versions, those on which all test agree. The prototype allows the analyst to browse the individual test results. This is especially useful when the test disagree on the set of possible OSes. Based on experiments in the lab where we used packet crafters and custom deception measures, we believe that an experience analyst can in general determine which test has failed and thus make an educated guess about the OS. In order to automate the management of discrepancies and corroborative elements, more effort would be required to enhance the correlation module. This would further increase the level of confidence in the fingerprinting techniques.

### 7.1.2 Network Conditions and Configuration

Tests that rely on matching stimulus and response and tests based on capturing samples of packets produced one after the other are sensitive to packet loss. For tests relying on samples, an effort was made to make the tests somewhat resilient to packet loss or packets arriving out of order. For tests based on stimulus and response, the problem arises only when the signature includes a check for responsiveness. Otherwise, the incomplete pair is simply discarded and thus will not produce a misleading result. In general, missing packets when doing passive fingerprinting is not as critical as in the context of intrusion detection. The fingerprinting device may get another chance to perform a test on subsequent packets. The intrusion detection system on the other hand is expected to remain effective at all times.

Packet loss may be caused by network congestion but may also be due to incompleteness of the network coverage. Improper coverage may not only alter performance of certain *Sample* and *Stimulus-Response* types of tests, but may also prevent hosts connected to the protected network from being detected. Network devices such as switches, routers and firewalls will limit any one sensor’s view of the entire network. Asymmetric routing topology and load balancing devices are particularly troublesome. Network Address Translation (NAT) technology makes all traffic from/to the protected network appear to be coming from or destined to one node, making difficult the identification of hosts behind the NATing device. Based on the topology and the coverage level desired, the locations and number of sensors required may vary greatly from one network to another. The optimum sensor deployment strategy is in general difficult to determine. It typically involves making decision to balance the cost, the load imposed on sensors, the ease of set-up and maintenance, and the monitoring coverage.

Two tests, PassiveTest_TCP_Timestamp and PassiveTest_ARP_Retransmit, are sensitive to delays. A variation in the round trip time due to network condition may alter the performance of those tests. Another limitation is due to the passive methodology itself. It may be impossible to determine whether a lack of response is attributed to configuration of the host, improper network coverage, or the presence of a filtering device. Moreover, the host not responding may be down or may not even exist. For a test based on determining whether or not a response was transmitted, the result is valid only in the first case (i.e. due to configuration of OS). Because the prototype currently has no mechanism to discriminate between these cases, tests based on responsiveness are likely to produce unsuitable results.

## 7.2 Future Work

This effort has focused on developing the individual tests and implementing those tests in a prototype capable of performing passive OS detection on live traffic or pre-recorded traffic traces. The prototype was designed to ease the signature learning process as well as the verification (evaluation) process. This allowed observing certain cases that had not been foreseen. Many tests would benefit from a number of adjustments as summarised in Table 11 of an upcoming section. We discuss here a couple of related research activities that would complement the work we have done.

A comprehensive analysis of the signature database would be an interesting follow-up of this work. For instance, the signature database could be analyzed with the intent of determining, when possible, the combination of tests required to isolate a given version of an operating system.

The signatures were collected from systems typically used as servers and workstations (PCs and laptop). In environments where computers are not locked-down[34], end-users are likely to change configuration and add or remove new systems without prior notice to network administrators. Because of this factor, the automated capability to discover operating systems of end nodes is particularly useful. Nonetheless, end-nodes are not the only TCP/IP stacks identified by fingerprinting tools. Routers, switches, printers, firewalls, web cameras, and even game consoles are identifiable. A desirable update for the database would include other types of networked components.

We are planning on using a similar passive fingerprinting approach to identify various virtual private network (VPN) implementations. A useful addition to the monitoring toolset would be the ability to identify through passive techniques the characteristics of VPN implementations such as IPSec and other similar protocols based on both proprietary and open source systems. It may be possible to identify specific implementations associated with versions of operating systems and implementations in network appliance devices.

The monitoring tools that have been developed by the Network Security Research group at CRC have been tested in a “wired” environment in the lab, and within some non-lab environments, using IPv4. With the potential rapid deployment of wireless

[34] Locked down computers have hardware or software configurations which prohibit users from modifying the configuration (e.g. preventing installation of new software).

LANs and other network technologies, such IPv6, including associated network services, the active/passive tools must be readied for use in these environments.  To be able to use these tools in a WLAN environment (802.11 a/b/g), some modifications may be necessary.  However, the experience gathered in building the tools and the algorithms used by the tools should be utilized in the wireless and IPv6 environments.

# 8. Conclusion

The Network Security Research Group at the Communication Research Centre (CRC) has developed a series of tests for passively detecting operating systems, and has implemented a prototype software tool as a proof of concept. For all the tests implemented in the passive tool, the approach taken was based on the analysis of packet headers at the data-link, network, and transport layers, thus the tool does not rely on access to application data. Over a dozen tests have been developed to analyse headers of packets seen on a network. The tests are conducted on headers of various protocols: ARP, IP, ICMP, UDP, and TCP. The prototype goes beyond analysis of individual packet commonly used in open source and commercial operating system identification tools. Because certain packets have influence on subsequent packets, some information can only be gained when related packets are analysed together. The use of lightweight state-aware mechanisms to derived signatures for operating systems is a unique approach. A test monitors the traffic for certain types of packets, produces a signature based on the values seen in the packets monitored, and does a lookup in a database to obtain a list of operating system associated with that signature.

The various tests developed were implemented in a JAVA prototype. On top of the individual tests, the prototype includes mechanisms that manage and combine the outcomes of all tests in order to produce the most likely subset of OS possibilities associated with an IP address. The prototype has different modes of operation allowing it to recognize new signatures, verify existing knowledge, or identify OSes of unknown computers on an unknown network. All modes can run on live traffic or pre-recorded traffic traces.

This document has described the OS fingerprinting techniques included in the passive operating system identification prototype. In section 2, we provided background on the state of the work on active and passive operating system identification. We gave a detailed description of the header fields of the IPv4 TCP/IP protocol suite that are useful in OS fingerprinting. In section 3, we described each of the individual tests developed for the prototype. As described in section 4, the signatures were collected from a private testbed. A variety of operating systems were installed and tested systematically in order to capture the signatures. The approach achieved uniformity in the testing of open source and non-open source operating systems. The signatures collected are included in Annex A. Section 5 discussed the mechanisms that combine the outcomes of individual tests together. Section 5 also presented the results obtained from a small scale monitoring experiment on a campus network. Certain signatures observed during this trial were different from those contained in the database. This was expected because of the difficulty of controlling elements of influence, such as application level software, when collecting the signatures. While the discrepancies sometimes prevented the program from finding perfect matches, the alternative signature lookup mechanisms were often able to correctly identify the OS nonetheless. Section 6 discussed the current state of the prototype and its integration to other monitoring tools developed at CRC. Lastly, limitations and desirable follow-up activities were described in section 7.

The tests are summarized in the following table, with their respective types and known problems and/or limitations.  This table also contains some ideas for enhancement.

***Table 11. Comments on each test***

| **Name of Test** | **Type** | **Comments** | **Known problems and proposed enhancements** |
|---|---|---|---|
| PassiveTest_TCP_SYN | Singleton | Almost identical to *p0f*'s SYN test in *p0f v2*. | We believe that the TCP Window size field and the value of the Window Scale TCP option of some OSes may vary depending on the application involved. The signatures were produced using a limited number of application clients. While the signatures give accurate results for most common services, we recommend that the signature collection process be revised to test a greater variety of application clients. Until then, the alternative signature mechanism helps in searching for non-perfect match. |
| PassiveTest_ARP_Requests | Singleton | ARP is not routed; therefore this test requires sharing a link-layer with the hosts under observation. | Based on traffic observed on a network different than the testbed, we believe that there could be more categories of possible values for the *Target Hardware address* than those anticipated. |
| PassiveTest_TCP_ISN | Sample | More reliable if all traffic transmitted by the host under observation can be captured. | Evaluation performed on real user traffic clearly indicates that the samples produced in the testbed to generate the signatures are insufficient to capture all possible responses. We recommend that the signature collection process be revised. More samples are required, and when possible, the TCP/IP stack implementation (program code) of open source OSes should be examined. Moreover, the algorithms that decide whether or not numbers are randomly generated are based on *nmap* program and are simplistic. Thus, the algorithms could be improved. |
| PassiveTest_TCP_TS | Sample | In contrast with other *Sample* test, this one can tolerate loss of packets. | As an enhancement, a subtest of type Singleton could be defined to observe the zeroing behaviour in SYN and SYN/ACK packets. PassiveTest_TCP_TS could then be modified to examine the update rate on **all** TCP packets. |

| Name of Test | Type | Comments | Known problems and proposed enhancements |
|---|---|---|---|
| PassiveTest_IP_ID | Sample | More reliable if all traffic transmitted by the host under observation can be captured. | As described in section 3.2.4, we observed unanticipated behaviour for Solaris and for Mac OS prior to Mac OS X.  The results (signatures) produced for these systems sometimes appear contradictory.  Improving the consistency of the classification algorithm would help improving the accuracy of this test. |
| subtest PassiveTest_Null_IP_ID | Singleton | This test removes from the sampling process of PassiveTest_IP_ID the packets having an IP ID of zero.  This test is also efficient at identifying certain OS. | Some OSes get two different signatures for a given type of packet.  In some cases the OS may send a null value on reboot, but nonzero values in subsequent packets.  In some other cases it is because the stimulus influences the IP ID value in the response (refer to section 3.2.4 for comments on TCP ACK packets transmitted by Linux in response to FIN/ACK packets).  A Singleton test cannot model such behaviour.  Perhaps the test type should be revisited to accommodate the few exceptional cases. |
| subtest PassiveTest_Echo_IP_ID | Stimulus-Response | Removes from the sampling process of PassiveTest_IP_ID the packets having an IP ID of zero.  Also efficient at identifying certain OS. | |

| **Name of Test** | **Type** | **Comments** | **Known problems and proposed enhancements** |
|---|---|---|---|
| PassiveTest_ARP_Retransmit | Sample | ARP requests are not routed. This test therefore requires sharing a link-layer with the target. This test is also sensitive to delays introduced by network congestion. | The current implementation listens for a sequence of identical ARP requests. The implementation assumes, without further checks, that these identical requests are retransmissions and that the ARP module itself produces these retransmissions. As described in section 3.2.6, this assumption does not always hold and so the test may produce false results. While monitoring only ARP requests provides a clear advantage since these packets are broadcasted, monitoring related packets (e.g. ARP replies) may prove to be useful. We recommend further investigation of the use of the *Stimulus-Response* matching algorithm to ensure that the sample consists only of unanswered ARP requests. |
| PassiveTest_ICMP_ID_SEQ | Sample | Requirements on the sample make this test less likely to be performed. | The database does not currently include signatures for most Novell systems. Further examination of Novell *ping* utility is required to understand the behaviour. Changes to the signature computation algorithm should then be made accordingly. |
| subtest PassiveTest_ICMP_SEQ | Sample | Has more chances to be performed than PassiveTest_ICMP_ID_SEQ. | |
| subtest PassiveTest_ICMP_ID | Singleton | Has more chances to be performed than PassiveTest_ICMP_ID_SEQ. | |

| Name of Test | Type | Comments | Known problems and proposed enhancements |
|---|---|---|---|
| PassiveTest_TCP_SYNACK | Stimulus-Response | This test reveals that taking the TCP options of the SYN into account helps gaining precision. This is what is lacking from most passive OS detection tools such as *p0f v2* and *ettercap*. | It appears that the WIN value of SYN/ACK produced by certain OSes may also depend on:<br>- the network service running (port),<br>- the window size advertised in the stimulus and<br>- the TCP window scale option of the stimulus.<br>We therefore recommend that the signature collection process be revised to test a greater variety of application services and that the signature calculation be modified to measure the influence of the items listed above. Moreover, when examining the WIN value, the program tries to determine if the value is a multiple of the MSS value (TCP option), if not it then examines if it is a multiple of the MSS advertised in the stimulus. It appears that the check should be done in reverse order. Refer to Table 20 for the comment concerning FreeBSD. This comment holds for Mac OS X, OpenBSD, Windows, SunOS 5.8 and 5.9, and NetBSD prior to 1.3 as well. |
| PassiveTest_TCP_RSTACK | Stimulus-Response | The influence of the SYN packet is clearly seen. Some OS versions echo certain fields in the RST/ACK. The program currently checks for echoing behaviour on the DF, TTL, WIN, and TCPopts fields. | |
| PassiveTest_ICMP_Unreach | Stimulus-Response | Performs well at providing small subsets of possible OS versions. | |

| **Name of Test** | **Type** | **Comments** | **Known problems and proposed enhancements** |
|---|---|---|---|
| PassiveTest_ICMP_Echo | Stimulus-Response | The stimulus being monitored is an abnormal packet, which makes this test less likely to be performed. | The test can be easily modified to obtain the same level of precision based on any pair of Echo Request/Reply messages. |
| PassiveTest_ICMP_Info | Stimulus-Response | It would be preferable to ensure that the target is up before concluding to a non-response from this host. | |
| PassiveTest_ICMP_Mask | Stimulus-Response | It would be preferable to ensure that the target is up before concluding to a non-response from this host. | |
| PassiveTest_ICMP_TS | Stimulus-Response | It would be preferable to ensure that the target is up before concluding to a non-response from this host. | |

# References

[1] *tcpdump* program, an open source tool for analysing packets originally developed at the Lawrence Berkeley National Lab. The tool can be downloaded from http://www.tcpdump.org.

[2] F. Veysset, O. Courtay, O. Heen, “New Tool and Technique for remote Operating System Fingerprinting”, April 2002. This document and the *ring* program were available originally at www.intranode.com/pdf/techno/. Any reference to *ring* has now disappeared from this site, however, a new version of *ring* (*ringv2*) can be obtained from http://ringv2.tuxfamily.org/index.html.

[3] Concept, “OS Detection with ARP”, Napalm e-zine, Issue 6, July 2000. Article can be found at http://www.defcon.tv/mag/napalm/napalm-6.txt. *induce-arp* (the tool implementing these techniques) can be downloaded from several sites, in particular from http://www.packetstormsecurity.org/UNIX/misc/induce-arp.tgz.

[4] F. Yarochkin, “Remote OS detection via TCP/IP Stack FingerPrinting”, October 18, 1998, available at www.insecure.org/nmap/nmap-fingerprinting-article.html. *Nmap* program is available at www.insecure.org /nmap/nmap_download.html.

[5] *queSO* program, a discontinued active OS fingerprinting tool by Savage, can still be downloaded from several sites, in particular from http://www.phreak.org/archives/ftp.cerias.purdue.edu/pub/tools/unix/scanners/queso/. The original homepage (http://www.apostols.org/) was teared down years ago.

[6] O. Arkin, F. Yarochkin, “X remote ICMP based OS fingerprinting techniques”, August 2001. The paper and *Xprobe* program can be download from http://www.sys-security.com/html/projects/X.html.

[7] O. Arkin, “ICMP Usage in Scanning”, June 2001, available at http://www.sys-security.com/html/projects/icmp.html

[8] nmap-hackers archive mailing list, http://lists.insecure.org/lists/nmap-hackers/. Communications are indexed per years.

[9] *p0f* program, a passive OS fingerprinting tool by Michal Zalewski and maintained by William Stearns. The tool can be downloaded from http://lcamtuf.coredump.cx/p0f.shtml

[10] *Ettercap* program, a multipurpose sniffer/interceptor/logger for switched LAN. It supports several active and passive features for network and host analysis. the tool can be downloaded from http://ettercap.sourceforge.net/.

[11] J. Nazario, “Passive System fingerprinting using Network Client Applications”, November 27, 2000, available at http://www.crimelabs.net/docs/passive.pdf

[12] P. Almquist, RFC 1349: “Type of Service in the Internet Protocol Suite”, status: proposed standard, July 1992, available at http://www.ietf.org/rfc/.

[13] K. Nichols, S. Blake, F. Baker, D. Black, RFC 2474: “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers”, status: proposed standard, December 1998, available at http://www.ietf.org/rfc/.

[14] K. Ramakrishnan, S. Floyd, D. Black RFC 3168: “The Addition of Explicit Congestion Notification (ECN) to IP”, status: proposed standard, September 2001, available at http://www.ietf.org/rfc/.

[15] J. Postel, RFC 791: “Internet Protocol”, status: standard, September 1981, available at http://www.ietf.org/rfc/.

[16] J. Postel, RFC 792: “Internet Control Message Protocol”, status: standard, September 1981, available at http://www.ietf.org/rfc/.

[17] W. Richard Stevens, “TCP/IP Illustrated”, Volume 1: The protocols, Addison-Wesley, 1994.

[18] J. Postel, RFC 793: “Transmission Control Protocol”, status: proposed standard, September 1981, available at http://www.ietf.org/rfc/.

[19] B. McDanel, Beyond Security Ltd, “TCP Timestamping - Obtaining System Uptime Remotely”, March 2001. Article can be found at the SecuriTeam.com web site http://www.securiteam.com/securitynews/5NP0C153PI.html

[20] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, RFC 2018: “TCP Selective Acknowledgement Options”, status: proposed standard, October 1996, available at http://www.ietf.org/rfc/.

[21] R. Braden, RFC 1122: “Requirements for Internet Hosts - Communication Layers”, status: standard, October 1989, available at http://www.ietf.org/rfc/.

[22] dethy, “Techniques To Validate Host-Connectivity”, whitepaper sent to bugtraq@securityfocus.com mailing list, January 2001. Archive available at http://lists.insecure.org/lists/bugtraq/2001/Jan/0218.html.

[23] VMWare’s Frequently Asked Questions (FAQ) website. Available at http://www.vmware.com/products/desktop/ws_faqs.html

[24] VirtualPC software from Connectix. Information about products is available at http://www.connectix.com/products/vpc5w.html

[25] Mac-on-Linux open-source software from Ibium HB. Information and downloads available at http://www.maconlinux.org

[26] *hping2* program, a packet crafter/analyser tool by Salvatore Sanfilippo, available at http://www.hping.org/

[27] F. Massicotte, T. Whalen and C. Bilodeau, “Network Mapping Tool for Real-Time Security Analysis”, NATO/RTO Symposium on Real-time Intrusion Detection, Lisbon Portugal, May 2002. Document available at ftp://ftp.rta.nato.int/PubFullText/RTO/MP/RTO-MP-101/MP-101-12.pdf

[28] Annie De Montigny-Leboeuf, Frédéric Massicotte, “Passive Network Discovery for Real Time Situation Awareness”, NATO/RTO Adaptive Defence in Unclassified Networks, Toulouse France, April 2004. Document available at ftp://ftp.rta.nato.int/PubFullText/RTO/MP/RTO-MP-IST-041/MP-IST-041-14.pdf

[29] Mathieu Couture, Béchir Ktari, Frédéric Massicotte, Mohamed Mejri, “A Declarative Approach to Stateful Intrusion Detection and Network Monitoring”, 2nd Annual Conference on Privacy, Security and Trust, Fredericton, New Brunswick, Canada, October 2004. Available at http://dev.hil.unb.ca/Texts/PST/pdf/couture.pdf

[30] *IP Personality*, a patch for Linux kernel 2.4 to impersonate TCP/IP stack of other operating systems, by Gaël Roualland, available at http://ippersonality.sourceforge.net/

[31] *Sygate Personal Firewall*, Product Information/Features and Benefits, http://smb.sygate.com/products/pspf/whatsnew_pspf.htm

[32] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP stack fingerprinting," in Proceedings of the 9th USENIX Security Symposium, August 2000. Available at http://www.usenix.org/publications/library/proceedings/sec2000/smart.html

[33] *pf.conf(5) Manual Page*, OpenBSD Programmer's Manual for the packet filter configuration file. OpenBSD Manual Pages can be browsed from http://www.openbsd.org/cgi-bin/man.cgi

# Annex A: Collected Signatures

This Annex provides the signatures collected from the testbed for all tests described in this document. As described in section 4.2, each test is concerned with two tables. One table contains the distinct signatures and in which each distinct signature is associated with a key identifier. The OS associations are stored in the other table. Each OS in this table is associated to one or more signatures by the mean of the key identifiers. The tables below are the results of queries made to that database, where OSes sharing a common signature have sometimes been regrouped to reduce the size of the tables. Footnotes have been added on some occasions to comment on certain peculiar behaviours.

***Table 12. PassiveTest_TCP_SYN***

| **PassiveTest_TCP_SYN** | | | | | |
|---|---|---|---|---|---|
| OS | DF | TTL | WIN | TCPecn | TCPopts |
| BEOS 5 | N | 255 | 12288 | | M@1460 |
| FreeBSD 2.0.5, 2.1.0 | N | 64 | 16384 | | M@1460NW@0NNTNNC.New |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | Y | 64 | 16384 | | M@1460NW@0NNTNNC.New |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | Y | 64 | 16384 | | M@1460 |
| FreeBSD 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 16384 | | M@1460 |
| FreeBSD 4.4 | Y | 64 | 16384 | | M@1460NW@0NNT |
| FreeBSD 4.5 | Y | 64 | 65535 | | M@1460NW@1NNT [35] |
| FreeBSD 4.6, 4.6.2, 4.7, 4.8 | Y | 64 | 57344 | | M@1460NW@0NNT |
| FreeBSD 5.0, 5.1 | Y | 64 | 65535 | | M@1460NW@1NNT |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 64 | 512 | | M@1460 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | Y | 64 | 22(MSS) | | M@1460STNW@0 |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | | M@1460STNW@0 |
| Linux 2.2.16 (S.u.S.E) | Y | 64 | 22(MSS) | | M@1460STNW@0 |
| Linux 2.2.16 (S.u.S.E) | Y | 64 | 32767 | | M@1460STNW@0 |
| Linux 2.2.18 (S.u.S.E) | Y | 64 | 22(MSS) | | M@1460STNW@0 |
| Linux 2.2.18 (S.u.S.E) | Y | 64 | 32767 | | M@1460STNW@0 |
| Linux 2.2.19 (Debian) | Y | 64 | 11(MSS) | | M@1460STNW@0 |
| Linux 2.2.20-idepci | Y | 64 | 11(MSS) | | M@1460STNW@0 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 4(MSS) | | M@1460STNW@0 |
| Linux 2.4.10-4GB | Y | 64 | 4(MSS) | | M@1460STNW@0 [36] |
| Linux 2.4.10-4GB | Y | 64 | 4(MSS) | | M@1460STNW@1 |

[35] FreeBSD 4.5 initiates a connection with Window size and Window scale values identical to what FreeBSD 5.0 and 5.1 use (destination port tested for all FreeBSD are 22, 23 and 21)

[36] Linux 2.4.10-4GB sends a Window scale of 0 from the telnet client but sends a Window scale of 1 using the ftp client.

| **PassiveTest_TCP_SYN** | | | | | |
|---|---|---|---|---|---|
| OS | DF | TTL | WIN | TCPecn | TCPopts |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 16616 | | M@1460W@0L |
| MacOS  9.0 | N | 255 | 32768 | | M@1460W@0N |
| MacOS  9.1, 9.2.1, 9.2.2 | Y | 255 | 32768 | | M@1460W@0N |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1,  10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 32768 | | M@1460NW@0NNT |
| NetBSD  1.1 | N | 64 | 16384 | | M@1460NW@0NNT |
| NetBSD  1.2, 1.2.1 | N | 64 | 16384 | | M@1460NW@0NNT |
| NetBSD  1.3 , 1.3.1, 1.3.2, 1.3.3 | N | 64 | 16384 | | M@1460NW@0NNT |
| NetBSD  1.4 , 1.4.1, 1.4.2, 1.4.3 | N | 64 | 16384 | | M@1460NW@0NNT |
| NetBSD  1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | | M@1460NW@0NNT |
| NetBSD  1.6, 1.6.1 | N | 64 | 16384 | | M@1460NW@0NNT@0 |
| Netware 4.11, 4.11 sp9 | N | 128 | 32768 | | M@1460 |
| Netware 5, 5 sp6a | Y | 128 | 32768 | | M@1460 |
| Netware 5.1, 5.1 sp6 | Y | 128 | 32768 | | M@1460 |
| Netware 6, 6 sp3 | Y | 128 | 6144 | | M@1460W@0NSNN |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 | N | 64 | 16384 | | M@1460NW@0NNT |
| OpenBSD  2.7, 2.8 | N | 64 | 16384 | | M@1460NNSNW@0NNT |
| OpenBSD  2.9 | Y | 64 | 16384 | | M@1460NNSNW@0NNT |
| OpenBSD  3.0, 3.1, 3.2, 3.3 | Y | 64 | 16384 | | M@1460NNSNW@0NNT |
| QNX RTP 4 | N | 64 | 8192 | | M@1460 |
| QNX RTP 6.0 | N | 64 | 8192 | | M@1459 |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | | M@1460NW@0NNT |
| SunOS  5.5, 5.5.1, 5.6, 5.7 | Y | 255 | 6(MSS) | | M@1460 |
| SunOS  5.8 | Y | 64 | 17(MSS) | | NNSM@1460 |
| SunOS  5.9 | Y | 64 | 34(MSS) | | M@1460NNS |
| SunOS (Intel) 5.8 | Y | 64 | 32850 | | NW@1NNTNNSM@1460 |
| Windows 95 | Y | 32 | 8192 | | M@1460 |
| Windows NT 3.51 standard | Y | 32 | 8192 | | M@1460 |
| Windows 98, 98 SE | Y | 128 | 8192 | | M@1460NNS |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 8192 | | M@1460 |
| Windows Millennium standard | Y | 128 | 16384 | | M@1460NNS |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 16384 | | M@1460NNS |
| Windows XP Home, Professional | Y | 128 | 16384 | | M@1460NNS |
| Windows Net standard | Y | 128 | 16384 | | M@1460NNS |
| Windows 2003 Server standard | Y | 128 | 16384 | | M@1460NNS |

***Table 13. PassiveTest_ARPRequest***

| **PassiveTest_ARPRequest** | |
|---|---|
| **OS** | **TargetHardwareAddress** |
| BEOS  5 | 000000000000 |
| FreeBSD  2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | 000000000000 |
| FreeBSD  3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | 000000000000 |

| PassiveTest_ARPRequest | |
|---|---|
| **OS** | **TargetHardwareAddress** |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5 | 000000000000 |
| FreeBSD 4.6, 4.6.2, 4.7, 4.8 | Uninitialized field |
| FreeBSD 5.0 | Uninitialized field |
| FreeBSD 5.1 | 000000000000 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | 000000000000 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | 000000000000 |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | 000000000000 |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | 000000000000 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | 000000000000 |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB,2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | 000000000000 |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | FFFFFFFFFFFF |
| MacOS 8.0, 8.1 | FFFFFFFFFFFF |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | FFFFFFFFFFFF |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | 000000000000 |
| NetBSD 1.1 | 000000000000 |
| NetBSD 1.2, 1.2.1 | 000000000000 |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | 000000000000 |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | 000000000000 |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | 000000000000 |
| NetBSD 1.6, 1.6.1 | 000000000000 |
| Netware 4.11, 4.11 sp9 | 000000000000 |
| Netware 5, 5 sp6a | 000000000000 |
| Netware 5.1, 5.1 sp6 | 000000000000 |
| Netware 6, 6 sp3 | 000000000000 |

| **PassiveTest_ARPRequest** | |
|---|---|
| **OS** | **TargetHardwareAddress** |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | 000000000000 |
| OpenBSD  3.0, 3.1, 3.2, 3.3 | 000000000000 |
| QNX RTP 4 | 000000000000 |
| QNX RTP 6.0, 6.1, 6.2, 6.2.1 | 000000000000 |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | FFFFFFFFFFFF |
| SunOS (Intel) 5.8 | FFFFFFFFFFFF |
| Windows 95 | 000000000000 |
| Windows NT 3.51 standard | 000000000000 |
| Windows 98, 98 SE | 000000000000 |
| Windows NT 4 standard, sp3, sp4, sp6 | 000000000000 |
| Windows Millennium standard | 000000000000 |
| Windows 2000 standard, sp2, sp3, sp4 | 000000000000 |
| Windows XP Home, Professional | 000000000000 |
| Windows Net standard | 000000000000 |
| Windows 2003 Server standard | 000000000000 |

***Table 14. PassiveTest_TCP_ISN***

| PassiveTest_TCP_ISN [37] | | | | | | |
|---|---|---|---|---|---|---|
| OS | class | val | gcdmin | gcdmax | stdmin | stdmax |
| BEOS  5 | RI | -1 | 1 | 1 | 1000892 | 2444429 |
| FreeBSD  2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1 | 64K | -1 | -1 | -1 | -1 | -1 |
| FreeBSD  2.2.0 | RI | -1 | 1 | 2 | 3929 | 53128 |
| FreeBSD  2.2.1 | RI | -1 | 1 | 1 | 4882 | 57589 |
| FreeBSD  2.2.2 | RI | -1 | 1 | 2 | 3207 | 52255 |
| FreeBSD  2.2.5 | RI | -1 | 1 | 3 | 6346 | 64159 |
| FreeBSD  2.2.6 | RI | -1 | 1 | 1 | 6910 | 59577 |
| FreeBSD  2.2.7 | RI | -1 | 1 | 2 | 6921 | 63422 |
| FreeBSD  2.2.8 | RI | -1 | 1 | 2 | 5568 | 60729 |
| FreeBSD  3.0 | RI | -1 | 1 | 2 | 6210 | 59938 |
| FreeBSD  3.1 | RI | -1 | 1 | 5 | 5559 | 62277 |

[37] The ranges for the gcd and std were estimated based on a limited number of samples (60 samples of 6 packets)

| PassiveTest_TCP_ISN [37] | | | | | | |
|---|---|---|---|---|---|---|
| OS | class | val | gcdmin | gcdmax | stdmin | stdmax |
| FreeBSD 3.2 | RI | -1 | 1 | 2 | 8698 | 63908 |
| FreeBSD 3.3 | RI | -1 | 1 | 2 | 7282 | 62903 |
| FreeBSD 3.4 | RI | -1 | 1 | 2 | 7262 | 67257 |
| FreeBSD 3.5.1 | RI | -1 | 1 | 3 | 6697 | 63850 |
| FreeBSD 4.0 | RI | -1 | 1 | 2 | 7260 | 62624 |
| FreeBSD 4.1 | RI | -1 | 1 | 2 | 5679 | 61199 |
| FreeBSD 4.1.1 | RI | -1 | 1 | 2 | 6355 | 61643 |
| FreeBSD 4.2 | RI | -1 | 1 | 3 | 10688 | 154539 |
| FreeBSD 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | TR | -1 | -1 | -1 | -1 | -1 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | TR | -1 | -1 | -1 | -1 | -1 |
| Linux 2.2.0 | RI | -1 | 1 | 2 | 635711 | 6294505 |
| Linux 2.2.1 | RI | -1 | 1 | 2 | 844449 | 6327873 |
| Linux 2.2.2 | RI | -1 | 1 | 2 | 442839 | 5885217 |
| Linux 2.2.3 | RI | -1 | 1 | 2 | 303081 | 5481445 |
| Linux 2.2.4 | RI | -1 | 1 | 3 | 1066440 | 5214589 |
| Linux 2.2.5 | RI | -1 | 1 | 2 | 1070076 | 6642435 |
| Linux 2.2.5-15 | RI | -1 | 1 | 1 | 659554 | 5742076 |
| Linux 2.2.6 | RI | -1 | 1 | 1 | 1187507 | 5427882 |
| Linux 2.2.7 | RI | -1 | 1 | 3 | 302863 | 5946898 |
| Linux 2.2.8 | RI | -1 | 1 | 1 | 1036204 | 5794169 |
| Linux 2.2.9 | RI | -1 | 1 | 2 | 535482 | 6612445 |
| Linux 2.2.10 | RI | -1 | 1 | 2 | 935261 | 5573627 |
| Linux 2.2.11 | RI | -1 | 1 | 3 | 693463 | 5549501 |
| Linux 2.2.12 | RI | -1 | 1 | 1 | 636419 | 6208814 |
| Linux 2.2.12-20 | RI | -1 | 1 | 9 | 198530 | 5763325 |
| Linux 2.2.13 | RI | -1 | 1 | 3 | 357923 | 6146175 |
| Linux 2.2.14 | RI | -1 | 1 | 2 | 440362 | 5341550 |
| Linux 2.2.14-5 | RI | -1 | 1 | 4 | 862625 | 6431236 |
| Linux 2.2.15 | RI | -1 | 1 | 3 | 1094033 | 6688537 |
| Linux 2.2.16 | RI | -1 | 1 | 3 | 418447 | 6335311 |
| Linux 2.2.16-22 | RI | -1 | 1 | 1 | 678021 | 5844348 |
| Linux 2.2.17 | RI | -1 | 1 | 2 | 790658 | 5953259 |
| Linux 2.2.18 | RI | -1 | 1 | 2 | 885130 | 6038753 |
| Linux 2.2.19 | RI | -1 | 1 | 1 | 648203 | 6106031 |
| Linux 2.2.20 | RI | -1 | 1 | 3 | 678499 | 6384234 |
| Linux 2.2.20-idepci | RI | -1 | 1 | 2 | 643181 | 6516181 |
| Linux 2.2.21 | RI | -1 | 1 | 2 | 807928 | 6511153 |
| Linux 2.2.22 | RI | -1 | 1 | 2 | 1365318 | 5387254 |
| Linux 2.2.23 | RI | -1 | 1 | 1 | 938740 | 5526417 |
| Linux 2.2.24 | RI | -1 | 1 | 4 | 624259 | 5624091 |
| Linux 2.4.0 | RI | -1 | 1 | 3 | 1145706 | 5697069 |
| Linux 2.4.1 | RI | -1 | 1 | 2 | 686064 | 5177326 |
| Linux 2.4.2 | RI | -1 | 1 | 2 | 814985 | 5910699 |
| Linux 2.4.2-2 | RI | -1 | 1 | 3 | 662334 | 6724921 |
| Linux 2.4.3 | RI | -1 | 1 | 2 | 1133825 | 5588171 |
| Linux 2.4.4 | RI | -1 | 1 | 2 | 747922 | 5608453 |
| Linux 2.4.4-4GB | RI | -1 | 1 | 3 | 633121 | 5372215 |
| Linux 2.4.5 | RI | -1 | 1 | 2 | 912120 | 5648993 |
| Linux 2.4.6 | RI | -1 | 1 | 2 | 872446 | 6468744 |
| Linux 2.4.7 | RI | -1 | 1 | 2 | 885387 | 5914056 |

| PassiveTest_TCP_ISN [37] | | | | | | |
|---|---|---|---|---|---|---|
| OS | class | val | gcdmin | gcdmax | stdmin | stdmax |
| Linux 2.4.8 | RI | -1 | 1 | 3 | 889712 | 6079950 |
| Linux 2.4.9 | RI | -1 | 1 | 2 | 761024 | 5587906 |
| Linux 2.4.10 | RI | -1 | 1 | 2 | 1101187 | 5697998 |
| Linux 2.4.10-4GB | RI | -1 | 1 | 3 | 956986 | 5914896 |
| Linux 2.4.11 | RI | -1 | 1 | 2 | 740901 | 5793945 |
| Linux 2.4.12 | RI | -1 | 1 | 3 | 1218711 | 5928584 |
| Linux 2.4.13 | RI | -1 | 1 | 2 | 1038365 | 6119539 |
| Linux 2.4.14 | RI | -1 | 1 | 1 | 1271446 | 6192395 |
| Linux 2.4.15 | RI | -1 | 1 | 2 | 1259864 | 5706854 |
| Linux 2.4.16 | RI | -1 | 1 | 2 | 969006 | 6284941 |
| Linux 2.4.17 | RI | -1 | 1 | 3 | 711902 | 5892498 |
| Linux 2.4.18 | RI | -1 | 1 | 5 | 223250 | 5586006 |
| Linux 2.4.18-14 | RI | -1 | 1 | 2 | 992302 | 5690466 |
| Linux 2.4.18-3 | RI | -1 | 1 | 2 | 899907 | 5863692 |
| Linux 2.4.18-4GB | RI | -1 | 1 | 3 | 1040789 | 6233509 |
| Linux 2.4.19 | RI | -1 | 1 | 3 | 962273 | 5899243 |
| Linux 2.4.19-4GB | RI | -1 | 1 | 4 | 583168 | 5255110 |
| Linux 2.4.20 | RI | -1 | 1 | 2 | 705151 | 5996601 |
| Linux 2.4.20-8 | RI | -1 | 1 | 2 | 872000 | 6366992 |
| Linux 2.4.21-0.13mdk | RI | -1 | 1 | 2 | 1018968 | 6074813 |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | 64K | -1 | -1 | -1 | -1 | -1 |
| MacOS 9.0 | RI | -1 | 1 | 2 | 6011 | 79545 |
| MacOS 9.1 | RI | -1 | 1 | 1 | 47162 | 81852 |
| MacOS 9.2.1 | RI | -1 | 1 | 1 | 4607 | 82016 |
| MacOS 9.2.2 | RI | -1 | 1 | 1 | 2996 | 76299 |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | TR | -1 | -1 | -1 | -1 | -1 |
| NetBSD 1.1, 1.2.1 | 64K | -1 | -1 | -1 | -1 | -1 |
| NetBSD 1.3 | RI | -1 | 1 | 2 | 39737 | 244693 |
| NetBSD 1.3.1 | RI | -1 | 1 | 4 | 23505 | 308482 |
| NetBSD 1.3.2 | RI | -1 | 1 | 2 | 30144 | 243626 |
| NetBSD 1.3.3 | RI | -1 | 1 | 2 | 36169 | 244680 |
| NetBSD 1.4 | RI | -1 | 1 | 1 | 3564039 | 16816537 |
| NetBSD 1.4.1 | RI | -1 | 1 | 2 | 2442106 | 17759148 |
| NetBSD 1.4.2 | RI | -1 | 1 | 2 | 2818277 | 17142013 |
| NetBSD 1.4.3 | RI | -1 | 1 | 4 | 1502736 | 19181040 |
| NetBSD 1.5 | RI | -1 | 1 | 2 | 866496 | 18067551 |
| NetBSD 1.5.1 | RI | -1 | 1 | 2 | 2396436 | 16758667 |
| NetBSD 1.5.2 | RI | -1 | 1 | 2 | 1675020 | 16614205 |
| NetBSD 1.5.3 | RI | -1 | 1 | 2 | 2740961 | 18160982 |
| NetBSD 1.6 | RI | -1 | 1 | 2 | 3287174 | 16747707 |
| NetBSD 1.6.1 | RI | -1 | 1 | 1 | 3757514 | 18888043 |

| PassiveTest_TCP_ISN [37] | | | | | | |
|---|---|---|---|---|---|---|
| OS | class | val | gcdmin | gcdmax | stdmin | stdmax |
| Netware 4.11 [38] | TD | -1 | 16 | 144 | 0 | 1 |
| Netware 4.11 sp9 | RI | -1 | 1 | 2 | 1077890 | 6767846 |
| Netware 5 | TD | -1 | 16 | 160 | 0 | 7 |
| Netware 5 sp6a | RI | -1 | 1 | 2 | 617559 | 8493698 |
| Netware 5.1 | RI | -1 | 1 | 2 | 942034 | 13029814 |
| Netware 5.1 sp6 | TR | -1 | -1 | -1 | -1 | -1 |
| Netware 6 | RI | -1 | 1 | 2 | 1106089 | 6497481 |
| Netware 6 sp3 | TR | -1 | -1 | -1 | -1 | -1 |
| OpenBSD 2.0 | RI | -1 | 1 | 2 | 6109 | 43773 |
| OpenBSD 2.1 | RI | -1 | 1 | 2 | 9275 | 40209 |
| OpenBSD 2.2 | RI | -1 | 1 | 4 | 6646 | 53004 |
| OpenBSD 2.3 | RI | -1 | 1 | 2 | 4214 | 39610 |
| OpenBSD 2.4 | RI | -1 | 1 | 2 | 7882 | 40040 |
| OpenBSD 2.5 | RI | -1 | 1 | 2 | 8002 | 40864 |
| OpenBSD 2.6 | RI | -1 | 1 | 2 | 13909 | 81834 |
| OpenBSD 2.7 | RI | -1 | 1 | 2 | 7323 | 79816 |
| OpenBSD 2.8 | RI | -1 | 1 | 2 | 19145 | 109835 |
| OpenBSD 2.9, 3.0, 3.1, 3.2, 3.3 | TR | -1 | -1 | -1 | -1 | -1 |
| QNX RTP 6.1 | RI | -1 | 1 | 2 | 53769 | 264995 |
| QNX RTP 6.2 | RI | -1 | 1 | 1 | 1581550 | 15103455 |
| QNX RTP 6.2.1 | RI | -1 | 1 | 3 | 1898378 | 15101966 |
| SunOS 5.5 | RI | -1 | 1 | 2 | 8607 | 47944 |
| SunOS 5.5.1 | RI | -1 | 1 | 1 | 10580 | 52878 |
| SunOS 5.6 | RI | -1 | 1 | 2 | 11790 | 51830 |
| SunOS 5.7 | RI | -1 | 1 | 2 | 13757 | 48936 |
| SunOS 5.8 | RI | -1 | 1 | 2 | 11240 | 79035 |
| SunOS 5.9 | RI | -1 | 1 | 2 | 15877 | 86686 |
| SunOS (Intel) 5.8 [39] | RI | -1 | 1 | 3 | 9599 | 1526560 |
| Windows 95 | TD | -1 | 1 | 2 | 1 | 31 |
| Windows 98 | TD | -1 | 1 | 2 | 0 | 3 |
| Windows 98 SE | TD | -1 | 1 | 3 | 5 | 141 |
| Windows NT 3.51 standard | TD | -1 | 1 | 3 | 0 | 306 |
| Windows NT 4 standard | TD | -1 | 1 | 1 | 0 | 39 |
| Windows NT 4 sp3 | TD | -1 | 1 | 10 | 0 | 29 |
| Windows NT 4 sp4 | TD | -1 | 1 | 2 | 1 | 5 |
| Windows NT 4 sp6 | TD | -1 | 1 | 1 | 2 | 8 |
| Windows NT 4 sp6 [40] | RI | -1 | 1 | 1 | 51195 | 51196 |
| Windows Millennium standard | RI | -1 | 1 | 2 | 3834 | 78789 |

[38] Service packs of Novell systems appear to influence the ISN generation. For Netware versions 4.11 and 5, the ISN class determined by the prototype went from Time-Dependent (TD) to Random Incremental (RI) when the service pack was added. Similarly, the class changed from RI to “true random” (TR) when a service pack was installed on Netware 5.1 and 6.

[39] ISN produced by Intel based SunOS may have a greater variability than those of Sparc based SunOS.

[40] 56 out of 60 samples produced a Time-Dependent (TD) signature for Windows NT 4 sp6. The Random Incremental (RI) signature is due to 4 samples, two with a std of 51195 and two with a std of 51196.

| PassiveTest_TCP_ISN [37] | | | | | | |
|---|---|---|---|---|---|---|
| OS | class | val | gcdmin | gcdmax | stdmin | stdmax |
| Windows 2000 Server sp2 | RI | -1 | 1 | 2 | 1253 | 16583 |
| Windows 2000 Server standard | RI | -1 | 1 | 2 | 2612 | 24155 |
| Windows 2000 sp2 | RI | -1 | 1 | 2 | 3233 | 23679 |
| Windows 2000 standard | RI | -1 | 1 | 2 | 1810 | 20529 |
| Windows 2000 sp3 | RI | -1 | 1 | 1 | 3240 | 26206 |
| Windows 2000 sp4 | RI | -1 | 1 | 2 | 1754 | 16946 |
| Windows XP Home | RI | -1 | 1 | 1 | 3253 | 17644 |
| Windows XP Professional | RI | -1 | 1 | 1 | 3770 | 12943 |
| Windows Net standard | TR | -1 | -1 | -1 | -1 | -1 |
| Windows 2003 Server standard | TR | -1 | -1 | -1 | -1 | -1 |

***Table 15. PassiveTest_Echo_IP_ID (Subtest of PassiveTest_IP_ID)***

| PassiveTest_Echo_IP_ID (Subtest of PassiveTest_IP_ID) | | | | | |
|---|---|---|---|---|---|
| ResultOSKey | IPIDEcho | Protocol | PacketType | StimulusProtocol | StimulusPacketType |
| QNX RTP 4 | Y | 1 | 0:0 | 1 | 8:0 |
| QNX RTP 6.0 | Y | 1 | 0:0 | 1 | 8:0 |

***Table 16. PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID)***

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 1 | 0:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 1 | 0:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 1 | 14:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 1 | 14:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 1 | 3:3 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 1 | 3:3 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 1 | 8:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 1 | 8:0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 17 | none |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 17 | none |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | A |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | AF |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | AP |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | AR |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 6 | AR |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | R |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 6 | R |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | S |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 6 | S |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 6 | SA |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | 6 | SA |

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 1 | 0:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 1 | 0:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 1 | 14:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 1 | 14:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 1 | 3:3 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 1 | 3:3 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 1 | 8:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 1 | 8:0 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 17 | none |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 17 | none |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | A |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | AF |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | AFP |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | AP |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | AR |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 6 | AR |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | R |

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 6 | R |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | S |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 6 | S |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 6 | SA |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 6 | SA |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 1 | 0:0 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 1 | 14:0 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 1 | 3:3 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 1 | 8:0 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | N | 17 | none |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 17 | none |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | A |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | AF |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | AFP |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | AP |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | AR |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | R |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | S |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3 | Y | 6 | SA |
| Linux 2.4.2-2 | Y | 1 | 0:0 |
| Linux 2.4.2-2 | Y | 1 | 14:0 |
| Linux 2.4.2-2 | N | 1 | 3:3 |
| Linux 2.4.2-2 | y | 1 | 3:3 |
| Linux 2.4.2-2 | Y | 1 | 8:0 |
| Linux 2.4.2-2 | N | 17 | none |
| Linux 2.4.2-2 | Y | 17 | none |
| Linux 2.4.2-2 | N | 6 | A |
| Linux 2.4.2-2 [41] | Y | 6 | A |
| Linux 2.4.2-2 | N | 6 | AF |
| Linux 2.4.2-2 | N | 6 | AFP |
| Linux 2.4.2-2 | N | 6 | AP |

[41] Linux 2.4.2-2 behaves differently than Linux 2.4.2. Linux 2.4.2-2 has two signatures for a TCP ACK packet: one with a null IPID and one with a non zero IP ID. This system sends a non zero IPID in a TCP ACK segment, unless this TCP ACK segment is the response to a FIN/ACK packet.

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux 2.4.2-2 | Y | 6 | AR |
| Linux 2.4.2-2 | Y | 6 | R |
| Linux 2.4.2-2 | N | 6 | S |
| Linux 2.4.2-2 | Y | 6 | SA |
| Linux 2.4.4, 2.4.4-4GB | Y | 1 | 0:0 |
| Linux 2.4.4, 2.4.4-4GB | Y | 1 | 14:0 |
| Linux 2.4.4, 2.4.4-4GB | N | 1 | 3:3 |
| Linux 2.4.4, 2.4.4-4GB [42] | Y | 1 | 3:3 |
| Linux 2.4.4, 2.4.4-4GB | Y | 1 | 8:0 |
| Linux 2.4.4, 2.4.4-4GB | N | 17 | none |
| Linux 2.4.4, 2.4.4-4GB | Y | 17 | none |
| Linux 2.4.4, 2.4.4-4GB | N | 6 | A |
| Linux 2.4.4, 2.4.4-4GB | N | 6 | AF |
| Linux 2.4.4, 2.4.4-4GB | N | 6 | AFP |
| Linux 2.4.4, 2.4.4-4GB | N | 6 | AP |
| Linux 2.4.4, 2.4.4-4GB | Y | 6 | AR |
| Linux 2.4.4, 2.4.4-4GB | Y | 6 | R |
| Linux 2.4.4, 2.4.4-4GB | N | 6 | S |
| Linux 2.4.4, 2.4.4-4GB | Y | 6 | SA |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 1 | 0:0 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 1 | 14:0 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 1 | 3:3 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 1 | 8:0 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 17 | none |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 17 | none |

[42] First packet after reboot. It appears that the IP ID of these kernels is incremantal for ICMP error messages (at least for port unreachable error messages). For these messages, the IP ID is globally incremented (e.g. 0x0000, 0x0100, 0x0200, etc) no matter what the destination is. The author has not seen the counter begin at 0 for TCP or UDP packets on reboot. Kernels 2.4.0-2.4.3 and 2.4.5-2.4.21 have a different behaviour. Kernels 2.4.0-2.4.3 use a null IP ID for all icmp port unreachable messages. Kernel 2.4.5-2.4.21 increment by 0x0001 instead of 0x0100 for icmp port unreach and do not appear to start with a zero value on reboot.

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 6 | A |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk [43] | Y | 6 | A |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 6 | AF |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 6 | AFP |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 6 | AP |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 6 | AR |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 6 | R |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | 6 | S |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.7-RH, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 6 | SA |
| Linux 2.4.19-4GB | N | 1 | 0:0 |
| Linux 2.4.19-4GB | Y | 1 | 0:0 |
| Linux 2.4.19-4GB | N | 1 | 14:0 |
| Linux 2.4.19-4GB | Y | 1 | 14:0 |
| Linux 2.4.19-4GB | N | 1 | 3:3 |
| Linux 2.4.19-4GB | Y | 1 | 3:3 |
| Linux 2.4.19-4GB | N | 1 | 8:0 |
| Linux 2.4.19-4GB | Y | 1 | 8:0 |
| Linux 2.4.19-4GB | N | 17 | none |
| Linux 2.4.19-4GB | Y | 17 | none |
| Linux 2.4.19-4GB | N | 6 | A |
| Linux 2.4.19-4GB | N | 6 | AF |
| Linux 2.4.19-4GB | N | 6 | AP |
| Linux 2.4.19-4GB | N | 6 | AR |
| Linux 2.4.19-4GB | Y | 6 | AR |
| Linux 2.4.19-4GB | N | 6 | R |
| Linux 2.4.19-4GB | Y | 6 | R |
| Linux 2.4.19-4GB | N | 6 | S |

[43] These systems sometimes send a null IPID in a TCP ACK packet transmitted in response to a FIN/ACK packet.

| PassiveTest_NULL_IP_ID (Subtest of PassiveTest_IP_ID) | | | |
|---|---|---|---|
| ResultOSKey | NullIPID | Protocol | PacketType |
| Linux   2.4.19-4GB | Y | 6 | S |
| Linux   2.4.19-4GB | N | 6 | SA |
| Linux   2.4.19-4GB | Y | 6 | SA |

***Table 17. PassiveTest_IP_ID***

| **PassiveTest_IP_ID** | | |
|---|---|---|
| **OS** | **IPIDClass** | **Protocol** |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | I-SI (or I) | -1 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | I-SI (or I) | 1 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | I-SI (or I) | 6 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | I-SI (or I) | 17 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | I-SI (or I) | -1 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | I-SI (or I) | 1 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | I-SI (or I) | 6 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | I-SI (or I) | 17 |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8,  5.0, 5.1 | I-SI (or I) | -1 |
| FreeBSD  4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8,  5.0, 5.1 | I-SI (or I) | 1 |
| FreeBSD  4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8,  5.0, 5.1 | I-SI (or I) | 6 |
| FreeBSD  4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8,  5.0, 5.1 | I-SI (or I) | 17 |
| Linux   2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | I-SI (or I) | -1 |
| Linux   2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | I-SI (or I) | 1 |
| Linux   2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | I-SI (or I) | 6 |
| Linux   2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | I-SI (or I) | 17 |
| Linux   2.2.0, 2.2.1, 2.2.2, 2.2.3,  2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | I-SI (or I) | -1 |
| Linux   2.2.0, 2.2.1, 2.2.2, 2.2.3,  2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | I-SI (or I) | 1 |

| PassiveTest_IP_ID | | |
|---|---|---|
| **OS** | **IPIDClass** | **Protocol** |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | I-SI (or I) | 6 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | I-SI (or I) | 17 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.3[44] | BI-SD (or BI) | 17 |
| Linux 2.4.2-2, 2.4.4, 2.4.4-4GB | I-SD (or I) | -1 |
| Linux 2.4.2-2, 2.4.4, 2.4.4-4GB | I-SD (or I) | 6 |
| Linux 2.4.2-2, 2.4.4, 2.4.4-4GB | I-SD (or I) | 17 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | I-SD (or I) | -1 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | I-SD (or I) | 1 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | I-SD (or I) | 6 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | I-SD (or I) | 17 |
| Linux 2.4.19-4GB[45] | I-SI (or I) | -1 |
| Linux 2.4.19-4GB | I-SI (or I) | 1 |
| Linux 2.4.19-4GB | I-SI (or I) | 6 |
| Linux 2.4.19-4GB | I-SI (or I) | 17 |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.2.2[46] | I-SD (or I-SI or I) | -1 |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.2.2 | I-SD (or I-SI or I) | 1 |

[44] Linux 2.4.0 to 2.4.3 have a null IPID otherwise

[45] Linux 2.4.19-4GB is the kernel of S.u.S.E 8.1. It distinguishes itself from other Linux kernels by incrementing the IP ID regardless of the socket.

[46] MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.2.2 get a signature with IPIDClass equal to “I-SD” if the sample contains communications with different IP destination addresses; they get a signature with IPIDClass equal to “I-SI” if the sample contains communications with the same IP destination address, but involving different sessions (e.g. a telnet and a ftp session running in parallel); they get a signature with IPIDClass equal to “I” if the sample contains one session with one end point.

| PassiveTest_IP_ID | | |
|---|---|---|
| OS | IPIDClass | Protocol |
| MacOS  7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.2.2 | I-SD (or I-SI or I) | 6 |
| MacOS  7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.2.2 | I-SD (or I-SI or I) | 17 |
| MacOS 10 10.0.0 | I-SI (or I) | -1 |
| MacOS 10 10.0.0 | I-SI (or I) | 1 |
| MacOS 10 10.0.0 | I-SI (or I) | 6 |
| MacOS 10 10.0.0 | I-SI (or I) | 17 |
| NetBSD  1.1, 1.2, 1.2.1, 1.3, 1.3.1, 1.3.2, 1.3.3, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | I-SI (or I) | -1 |
| NetBSD  1.1, 1.2, 1.2.1, 1.3, 1.3.1, 1.3.2, 1.3.3, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | I-SI (or I) | 1 |
| NetBSD  1.1, 1.2, 1.2.1, 1.3, 1.3.1, 1.3.2, 1.3.3, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | I-SI (or I) | 6 |
| NetBSD  1.1, 1.2, 1.2.1, 1.3, 1.3.1, 1.3.2, 1.3.3, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | I-SI (or I) | 17 |
| Netware 4.11, 4.11 sp9, 5, 5 sp6a, 5.1 | I-SI (or I) | -1 |
| Netware 4.11, 4.11 sp9, 5, 5 sp6a, 5.1 | I-SI (or I) | 1 |
| Netware 4.11, 4.11 sp9, 5, 5 sp6a, 5.1 | I-SI (or I) | 6 |
| Netware 4.11, 4.11 sp9, 5, 5 sp6a, 5.1 | I-SI (or I) | 17 |
| Netware 5.1 sp6, 6, 6 sp3 | BI-SI (or BI) | -1 |
| Netware 5.1 sp6, 6, 6 sp3 | BI-SI (or BI) | 1 |
| Netware 5.1 sp6, 6, 6 sp3 | BI-SI (or BI) | 6 |
| Netware 5.1 sp6, 6, 6 sp3 | BI-SI (or BI) | 17 |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4 | I-SI (or I) | -1 |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4 | I-SI (or I) | 1 |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4 | I-SI (or I) | 6 |
| OpenBSD  2.0, 2.1, 2.2, 2.3, 2.4 | I-SI (or I) | 17 |
| OpenBSD  2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | RD | -1 |
| OpenBSD  2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | RD | 1 |
| OpenBSD  2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | RD | 6 |
| OpenBSD  2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | RD | 17 |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | I-SI (or I) | -1 |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | I-SI (or I) | 1 |

| PassiveTest_IP_ID | | |
|---|---|---|
| **OS** | **IPIDClass** | **Protocol** |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | I-SI (or I) | 6 |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | I-SI (or I) | 17 |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.8(Intel), 5.9 | I-SD (or I-SI or I) | -1 |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.8(Intel), 5.9 | I-SD (or I-SI or I) | 1 |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.8(Intel), 5.9 | I-SD (or I-SI or I) | 6 |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.8(Intel), 5.9 | I-SD (or I) | 17 |
| Windows 95, NT 3.51 standard, 98, 98 SE | BI-SI (or BI) | -1 |
| Windows 95, NT 3.51 standard, 98, 98 SE | BI-SI (or BI) | 1 |
| Windows 95, NT 3.51 standard, 98, 98 SE | BI-SI (or BI) | 6 |
| Windows 95, NT 3.51 standard, 98, 98 SE | BI-SI (or BI) | 17 |
| Windows NT 4 standard, sp3, sp4, sp6 | BI-SI (or BI) | -1 |
| Windows NT 4 standard, sp3, sp4, sp6 | BI-SI (or BI) | 1 |
| Windows NT 4 standard, sp3, sp4, sp6 | BI-SI (or BI) | 6 |
| Windows NT 4 standard, sp3, sp4, sp6 | BI-SI (or BI) | 17 |
| Windows Millennium standard | I-SI (or I) | -1 |
| Windows Millennium standard | I-SI (or I) | 1 |
| Windows Millennium standard | I-SI (or I) | 6 |
| Windows Millennium standard | I-SI (or I) | 17 |
| Windows 2000 standard, sp2, sp3, sp4 | I-SI (or I) | -1 |
| Windows 2000 standard, sp2, sp3, sp4 | I-SI (or I) | 1 |
| Windows 2000 standard, sp2, sp3, sp4 | I-SI (or I) | 6 |
| Windows 2000 standard, sp2, sp3, sp4 | I-SI (or I) | 17 |
| Windows XP Home, Professional | I-SI (or I) | -1 |
| Windows XP Home, Professional | I-SI (or I) | 1 |
| Windows XP Home, Professional | I-SI (or I) | 6 |
| Windows XP Home, Professional | I-SI (or I) | 17 |

| PassiveTest_IP_ID | | |
|---|---|---|
| **OS** | **IPIDClass** | **Protocol** |
| Windows Net standard | I-SI (or I) | -1 |
| Windows Net standard | I-SI (or I) | 1 |
| Windows Net standard | I-SI (or I) | 6 |
| Windows Net standard | I-SI (or I) | 17 |
| Windows 2003 Server standard | I-SI (or I) | -1 |
| Windows 2003 Server standard | I-SI (or I) | 1 |
| Windows 2003 Server standard | I-SI (or I) | 6 |
| Windows 2003 Server standard | I-SI (or I) | 17 |

***Table 18. PassiveTest_TCP_TS***

| PassiveTest_TCP_TS | |
|---|---|
| **OS** | **TSClass** |
| FreeBSD  2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | 2HZ |
| FreeBSD  4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | 100HZ |
| Linux   2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | 100HZ |
| Linux   2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | 100HZ |
| Linux  2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | 100HZ |
| Linux   2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | 100HZ |
| Linux   2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB | 100HZ |
| Linux   2.4.18-14 | 500HZ |
| Linux   2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | 100HZ |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | 1000HZ |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1,  10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | 2HZ |
| NetBSD  1.1 | 2HZ |

| PassiveTest_TCP_TS | |
|---|---|
| **OS** | **TSClass** |
| NetBSD 1.2, 1.2.1 | 2HZ |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | 2HZ |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | 2HZ |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | 2HZ |
| NetBSD 1.6, 1.6.1 | Z |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | 2HZ |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | 2HZ |
| QNX RTP 6.2, 6.2.1 | 2HZ |
| SunOS 5.6, 5.7, 5.8, 5.9 | 100HZ |
| SunOS (Intel) 5.8 | 100HZ |
| Windows 2000 standard, sp2, sp3, sp4 | Z |
| Windows Millennium standard | Z |
| Windows XP Home, Professional | Z |
| Windows Net standard | Z |
| Windows 2003 Server standard | Z |

***Table 19. PassiveTest_ARPRetransmit***

| PassiveTest_ARPRetransmit | | | | |
|---|---|---|---|---|
| OS | NbOfPackets | DelayMin | DelayMax | ARPClass |
| BEOS 5 | 1 | | | SP |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | 1 | | | SP |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | 1 | | | SP |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | 1 | | | SP |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | infinit [47] | 5000000 | 60000000 | UNKNOWN |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3 | 4 | 1000000 | 1000000 | C |
| Linux 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | 3 | 1000000 | 1000000 | C |

[47] Linux 2.0.29 to 2.0.36 keep retransmitting unaswered ARP requests. The first 4 packets are separated by 5 seconds, then all remaining packets are separated by 60 seconds.

## PassiveTest_ARPRetransmit

| OS | NbOfPackets | DelayMin | DelayMax | ARPClass |
|---|---|---|---|---|
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | 3 | 1000000 | 1000000 | C |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | 3 | 1000000 | 1000000 | C |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | 3 | 1000000 | 1000000 | C |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB,2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | 3 | 1000000 | 1000000 | C |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | 6 | 1000000 | 1000000 | C |
| MacOS 8.0, 8.1 | 6 | 1000000 | 1000000 | C |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | 6 | 1000000 | 1000000 | C |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | 1 |  |  | SP |
| NetBSD 1.1 | 1 |  |  | SP |
| NetBSD 1.2, 1.2.1 | 1 |  |  | SP |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | 1 |  |  | SP |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | 1 |  |  | SP |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | 1 |  |  | SP |
| NetBSD 1.6, 1.6.1 | 1 |  |  | SP |
| Netware 4.11, 4.11 sp9, | 1 |  |  | SP |
| Netware 5, 5 sp6a | 1 |  |  | SP |
| Netware 5.1, 5.1 sp6 | 1 |  |  | SP |
| Netware 6, 6 sp3 | 1 |  |  | SP |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | 1 |  |  | SP |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | 1 |  |  | SP |
| QNX RTP 4 | 1 |  |  | SP |
| QNX RTP 6.0, 6.1, 6.2, 6.2.1 | 1 |  |  | SP |
| SunOS 5.5, 5.5.1 | 6 | 900000 | 900000 | C |
| SunOS 5.6, 5.7, 5.8, 5.9, (Intel) 5.8 | 6 | 1000000 | 1000000 | C |
| Windows 95 | 1 |  |  | SP |
| Windows NT 3.51 standard | 1 |  |  | SP |
| Windows 98, 98 SE | 1 |  |  | SP |
| Windows NT 4 standard, sp3, sp4, sp6 | 1 |  |  | SP |
| Windows Millennium standard | 1 |  |  | SP |
| Windows 2000 standard, sp2, sp3, sp4 | 1 |  |  | SP |
| Windows XP Home, Professional | 1 |  |  | SP |
| Windows Net standard | 1 |  |  | SP |
| Windows 2003 Server standard | 1 |  |  | SP |

*Table 20. PassiveTest_TCP_SYNACK*

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {C.NewM@1460TW} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1459} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1460} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1460S} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1460STW} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1460T@0W} |
| BEOS 5 | N | 255 | 12288 | S++ | | M@1460 | | {M@1460TW} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 17280 | S++ | | M@1460NW@0NNTNNCNNC.Echo | | {C.NewM@1460TW} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 12(MSSReq)[48] | S++ | | M@1460 | | {M@1459} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| FreeBSD 2.1.5, 2.1.6, 2.1.7.1, 2,2,0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460TW} |

[48] 12(MSSReq) means that the TCP Window Size (WIN) in the response is equal to twelve times the TCP Maximum Segment Size (MSS) advertized in the request (i.e. the SYN packet). In this particular signature 12(MSSReq) means that the WIN was equal to 12x1459=17508. It can be infer from this signature that when the WIN value in FreeBSD SYN/ACK is related to a MSS value, the influence comes from the MSS advertized in the SYN rather than from the MSS value advertized in the SYN/ACK response. This observation holds for Mac OS X, OpenBSD, Windows, SunOS 5.8 and 5.9, and NetBSD prior to 1.3.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {C.NewM@1460TW} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460STW} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460T@0W} |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1, 4.0, 4.1, 4.1.1,4.2, 4.3 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460TW} |
| FreeBSD 4.4 | Y | 64 | 17376 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| FreeBSD 4.4 | Y | 64 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| FreeBSD 4.4 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| FreeBSD 4.4 | Y | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| FreeBSD 4.4 | Y | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| FreeBSD 4.4 | Y | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| FreeBSD 4.4 | Y | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460TW} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460NW@1NNT | | {C.NewM@1460TW} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460 | | {M@1459} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460 | | {M@1460} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460 | | {M@1460S} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460NW@1NNT | | {M@1460STW} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460NW@1NNT | | {M@1460T@0W} |
| FreeBSD 4.5 | N | 64 | 65535 | S++ | | M@1460NW@1NNT | | {M@1460TW} |
| FreeBSD 4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460 |  | {M@1459} |
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460 |  | {M@1460} |
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460 |  | {M@1460S} |
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460NW@0NNT |  | {M@1460STW} |
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460NW@0NNT |  | {M@1460T@0W} |
| FreeBSD  4.6, 4.6.2, 4.7, 4.8 | N | 64 | 57344 | S++ |  | M@1460NW@0NNT |  | {M@1460TW} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460NW@1NNT |  | {C.NewM@1460TW} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460 |  | {M@1459} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460 |  | {M@1460} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460 |  | {M@1460S} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460NW@1NNT |  | {M@1460STW} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460NW@1NNT |  | {M@1460T@0W} |
| FreeBSD  5.0, 5.1 | Y | 64 | 65535 | S++ |  | M@1460NW@1NNT |  | {M@1460TW} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1459 |  | {M@1459} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {M@1460} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {M@1460S} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {M@1460STW} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {M@1460T@0W} |
| Linux   2.0.29 (Debian) | N | 64 | 15360 | S++ |  | M@1460 |  | {M@1460TW} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1459 |  | {M@1459} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {M@1460} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {M@1460S} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {M@1460STW} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {M@1460T@0W} |
| Linux   2.0.30 (RedHat) | N | 64 | 31744 | S++ |  | M@1460 |  | {M@1460TW} |
| Linux   2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Linux   2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ |  | M@1459 |  | {M@1459} |
| Linux   2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ |  | M@1460 |  | {M@1460} |
| Linux   2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ |  | M@1460 |  | {M@1460S} |
| Linux   2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ |  | M@1460 |  | {M@1460STW} |

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| Linux 2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ | | M@1460 | | {M@1460T@0W} |
| Linux 2.0.32, 2.0.36 (RedHat) | N | 64 | 32736 | S++ | | M@1460 | | {M@1460TW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1459 | | {M@1459} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1459 | | {M@1459} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460S} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460S} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460STW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460STW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460T@0W} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460T@0W} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460TW} |
| Linux 2.0.34, 2.0.36 (Debian) | N | 64 | 16352 | S++ | | M@1460 | | {M@1460TW} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ | | M@1460NNTNW@0 | | {C.NewM@1460TW} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ | | M@1459 | | {M@1459} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ | | M@1460 | | {M@1460} |

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ |  | M@1460NNS |  | {M@1460S} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ |  | M@1460STNW@0 |  | {M@1460STW} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ |  | M@1460NNTNW@0 |  | {M@1460T@0W} |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | 64 | 22(MSS) | S++ |  | M@1460NNTNW@0 |  | {M@1460TW} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460NNTNW@0 |  | {C.NewM@1460TW} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1459 |  | {M@1459} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460 |  | {M@1460} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460NNS |  | {M@1460S} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460STNW@0 |  | {M@1460STW} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460NNTNW@0 |  | {M@1460T@0W} |
| Linux 2.2.19, 2.2.20-idepci (Debian) | Y | 64 | 11(MSS) | S++ |  | M@1460NNTNW@0 |  | {M@1460TW} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 5792 | S++ |  | M@1460NNTNW@0 |  | {C.NewM@1460TW} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 4(MSS) | S++ |  | M@1460 |  | {M@1459} |

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 4(MSS) | S++ | | M@1460 | | {M@1460} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 4(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 5792 | S++ | | M@1460STNW@0 | | {M@1460STW} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 4(MSS) | S++ | | M@1460NW@0 [49] | | {M@1460T@0W} |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9, 2.4.10, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 5792 | S++ | | M@1460NNTNW@0 | | {M@1460TW} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ | | M@1460W@0L | | {C.NewM@1460TW} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 13(MSS) | S++ | | M@1459 | | {M@1459} |

[49] Linux 2.4.0 and above stop supporting the TCP Timestamp option if TSval is set to zero (T@0) in the TCP timestamp option of the SYN request.

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460 |  | {M@1460} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460 |  | {M@1460S} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460 [50] |  | {M@1460STW} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {M@1460STW} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {M@1460T@0W} |
| MacOS 7.5.3, 7.5.5 | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {M@1460TW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 [51] | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {C.NewM@1460TW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 65535 | S++ |  | M@1460W@2L |  | {C.NewM@1460TW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 13(MSS) | S++ |  | M@1459 |  | {M@1459} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 44(MSS) | S++ |  | M@1459 |  | {M@1459} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ |  | M@1460 |  | {M@1460} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 44(MSS) | S++ |  | M@1460 |  | {M@1460} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ |  | M@1460 |  | {M@1460S} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 44(MSS) | S++ |  | M@1460 |  | {M@1460S} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ |  | M@1460 |  | {M@1460STW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {M@1460STW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 44(MSS) | S++ |  | M@1460 |  | {M@1460STW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 65535 | S++ |  | M@1460W@2L |  | {M@1460STW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ |  | M@1460W@0L |  | {M@1460T@0W} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 65535 | S++ |  | M@1460W@2L |  | {M@1460T@0W} |

[50] MacOS 7 to 8 have two signatures for the SYN stimulus with SYN_SetOfTCPopts={M@1460STW@0}. Recall that the TCP options in *SYN_SetOfTCPopts* are listed in alphabetic order while the TCP options in *TCPops* are listed in the order they appear. MacOS 7 to 8 are the only systems we have seen for which the order in which the TCP options appear in the SYN influence the options advertised in response. Experiments conducted on the testbed lead us to believe that these systems do not process the last TCP option appearing in the SYN packet. Aside from the NOP and the EOL options, these machines only support the MSS and Window scale option. They show support for both options when stimulated by OpenBSD 2.9 (SYN_SetOfTCPopts={M@1460STW@0}) for which the options appear in the following order M@1460NNSNW@0NNT. The Window Scale option is missing from their response to Linux 2.4.7 (SYN_SetOfTCPopts={M@1460STW@0}) for which the options appear in the following order M@1460STNW@0.

[51] The Window size (WIN) value in SYN/ACK of MacOS 7.6 to 8.1 could be influence by the service running and the WIN value advertized in the stimulus (omitted from the signature).

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 12(MSS) | S++ | | M@1460W@0L | | {M@1460TW} |
| MacOS 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 65535 | S++ | | M@1460W@2L | | {M@1460TW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 [52] | N | 255 | 32768 | S++ | | M@1460W@0NNNT | | {C.NewM@1460TW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460W@2NNNT | | {C.NewM@1460TW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1459 | | {M@1459} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1459 | | {M@1459} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1460 | | {M@1460} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460 | | {M@1460} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1460 | | {M@1460S} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460 | | {M@1460S} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1460W@0NNNT | | {M@1460STW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460W@2NNNT | | {M@1460STW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1460W@0NNNT | | {M@1460T@0W} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460W@2NNNT | | {M@1460T@0W} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 32768 | S++ | | M@1460W@0NNNT | | {M@1460TW} |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | 255 | 65535 | S++ | | M@1460W@2NNNT | | {M@1460TW} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 [53] | Y | 64 | 33304 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 23(MSSReq)[54] | S++ | | M@1460 | | {M@1459} |

[52] The Window size (WIN) value in SYN/ACK of MacOS 9 could be influence by the service running and the WIN value advertized in the stimulus (omitted from the signature).

[53] The Window size (WIN) value in SYN/ACK of MacOS X appears to be independent of service running, but to depend on the presence of Window Scale option in the stimulus.

[54] “MSSReq” means that the TCP Window Size (WIN) in the response is related to TCP Maximum Segment Size (MSS) advertized in the request (i.e. M@1459 of the SYN packet). It can be infer from this signature that when the WIN value in MacOS X’s SYN/ACK is related to a MSS value, the influence comes from the MSS advertized in the SYN rather than from the MSS value advertized in the SYN/ACK response.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 23(MSS) | S++ | | M@1460 | | {M@1460} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 23(MSS) | S++ | | M@1460 | | {M@1460S} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 33304 | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 33304 | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | 64 | 33304 | S++ | | M@1460NW@0NNT | | {M@1460TW} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSSReq)[55] | S++ | | M@1460 | | {M@1459} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| NetBSD 1.1, 1.2, 1.2.1 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460TW} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ | | M@1460 | | {M@1459} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ | | M@1460 | | {M@1460} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ | | M@1460 | | {M@1460S} |

[55] "MSSReq" means that the TCP Window Size (WIN) in the response is related to TCP Maximum Segment Size (MSS) advertized in the request (i.e. M@1459 of the SYN packet). It can be infer from this signature that when the WIN value in a SYN/ACK transmitted by NetBSD 1.1 and 1.2 is related to a MSS value, the influence comes from the MSS advertized in the SYN rather than from the MSS value advertized in the SYN/ACK response.

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT |  | {M@1460STW} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT |  | {M@1460T@0W} |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3, 1.4 , 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT |  | {M@1460TW} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT@0 |  | {C.NewM@1460TW} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460 |  | {M@1459} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460 |  | {M@1460} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460 |  | {M@1460S} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT@0 |  | {M@1460STW} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT@0 |  | {M@1460T@0W} |
| NetBSD 1.6, 1.6.1 | N | 64 | 16384 | S++ |  | M@1460NW@0NNT@0 |  | {M@1460TW} |
| Netware 4.11 [56] | N | 128 | 2000 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Netware 4.11 | N | 128 | 32768 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Netware 4.11 | N | 128 | 65535 | S++ |  | M@1460 |  | {C.NewM@1460TW} |
| Netware 4.11 | N | 128 | 32768 | S++ |  | M@1459 |  | {M@1459} |
| Netware 4.11 | N | 128 | 65535 | S++ |  | M@1459 |  | {M@1459} |
| Netware 4.11 | N | 128 | 2000 | S++ |  | M@1460 |  | {M@1460} |
| Netware 4.11 | N | 128 | 32768 | S++ |  | M@1460 |  | {M@1460} |
| Netware 4.11 | N | 128 | 65535 | S++ |  | M@1460 |  | {M@1460} |
| Netware 4.11 | N | 128 | 2000 | S++ |  | M@1460 |  | {M@1460S} |
| Netware 4.11 | N | 128 | 32768 | S++ |  | M@1460 |  | {M@1460S} |
| Netware 4.11 | N | 128 | 65535 | S++ |  | M@1460 |  | {M@1460S} |
| Netware 4.11 | N | 128 | 2000 | S++ |  | M@1460 |  | {M@1460STW} |
| Netware 4.11 | N | 128 | 32768 | S++ |  | M@1460 |  | {M@1460STW} |
| Netware 4.11 | N | 128 | 65535 | S++ |  | M@1460 |  | {M@1460STW} |

[56] Examination of the traffic traces indicate that the WIN value of Netware 4.11 is influence by the network service running.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| Netware 4.11 | N | 128 | 32768 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 4.11 | N | 128 | 65535 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 4.11 | N | 128 | 2000 | S++ | | M@1460 | | {M@1460TW} |
| Netware 4.11 | N | 128 | 32768 | S++ | | M@1460 | | {M@1460TW} |
| Netware 4.11 | N | 128 | 65535 | S++ | | M@1460 | | {M@1460TW} |
| Netware 4.11 sp9 [57] | Y | 128 | 6144 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1459 | | {M@1459} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460S} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460STW} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 4.11 sp9 | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5 [58] | Y | 128 | 32768 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5 | Y | 128 | 8191 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1459 | | {M@1459} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1459 | | {M@1459} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1460 | | {M@1460} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460} |
| Netware 5 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1460 | | {M@1460S} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460S} |
| Netware 5 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460S} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 5 | Y | 128 | 32768 | S++ | | M@1460 | | {M@1460TW} |

[57] The service pack sp9 of Netware 4.11 seems to prevent variation in the WIN value.

[58] Examination of the traffic traces indicate that the WIN value of Netware 5 is influence by the network service running.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| Netware 5 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5 sp6a [59] | Y | 128 | 6144 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1459 | | {M@1459} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460S} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 5 sp6a | Y | 128 | 6144 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5.1 [60] | Y | 128 | 65535 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {C.NewM@1460TW} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1459 | | {M@1459} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1459 | | {M@1459} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460S} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460S} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460STW} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460T@0W} |
| Netware 5.1 | Y | 128 | 65535 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5.1 | Y | 128 | 8191 | S++ | | M@1460 | | {M@1460TW} |
| Netware 5.1 sp6, 6, 6 sp3 [61] | Y | 128 | 6144 | S++ | | M@1460W@0N | | {C.NewM@1460TW} |

[59] The service pack sp6a of Netware 5 seems to prevent variation in the WIN value.

[60] Examination of the traffic traces indicate that the WIN value of Netware 5.1 is influence by the network service running.

[61] The service pack sp6 of Netware 5.1 seems to prevent variations in the WIN value.  The WIN value of Netware 6 does not seem to be influence by the network service running whether a service pack is installed or not.

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1459 |  | {M@1459} |
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1460 |  | {M@1460} |
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1460SNN |  | {M@1460S} |
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1460W@0NSNN |  | {M@1460STW} |
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1460W@0N |  | {M@1460T@0W} |
| Netware 5.1 sp6, 6, 6 sp3 | Y | 128 | 6144 | S++ |  | M@1460W@0N |  | {M@1460TW} |

PassiveTest_TCP_SYNACK

| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
|---|---|---|---|---|---|---|---|---|
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSSReq)[62] | S++ | | M@1460 | | {M@1459} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4 | N | 64 | 12(MSS) | S++ | | M@1460NW@0NNT | | {M@1460TW} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {C.NewM@1460TW} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460S} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {M@1460STW} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {M@1460T@0W} |
| OpenBSD 2.5, 2.6 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {M@1460TW} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {C.NewM@1460TW} |
| OpenBSD 2.7 | N | 64 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1448NNSNW@0NNT | | {M@1460STW} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {M@1460T@0W} |
| OpenBSD 2.7 | N | 64 | 12(MSS) | S++ | | M@1448NW@0NNT | | {M@1460TW} |
| OpenBSD 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| OpenBSD 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| OpenBSD 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 12(MSS) | S++ | | M@1460 | | {M@1460} |

[62] “MSSReq” means that the TCP Window Size (WIN) in the response is related to TCP Maximum Segment Size (MSS) advertized in the request (i.e. M@1459 of the SYN packet). It can be infer from this signature that when the WIN value in a SYN/ACK transmitted by OpenBSD is related to a MSS value, the influence comes from the MSS advertized in the SYN rather than from the MSS value advertized in the SYN/ACK response.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| OpenBSD  2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| OpenBSD  2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 17376 | S++ | | M@1460NNSNW@0NNT | | {M@1460STW} |
| OpenBSD  2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| OpenBSD  2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | N | 64 | 17376 | S++ | | M@1460NW@0NNT | | {M@1460TW} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {C.NewM@1460TW} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1459} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460S} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460STW} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460T@0W} |
| QNX RTP 4 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460TW} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {C.NewM@1460TW} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1459} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1459 [63] | | {M@1460} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1459 | | {M@1460S} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460S} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1459 | | {M@1460STW} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460STW} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460T@0W} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1459 | | {M@1460TW} |
| QNX RTP 6.0 | N | 64 | 8192 | S++ | | M@1460 | | {M@1460TW} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460NW@0NNT | | {C.NewM@1460TW} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460 | | {M@1459} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460 | | {M@1460} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460 | | {M@1460S} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460NW@0NNT | | {M@1460STW} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460NW@0NNT | | {M@1460T@0W} |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 16384 | S++ | | M@1460NW@0NNT | | {M@1460TW} |

[63] QNX 6.0 sometimes advertize a Mazimum Segment Size (MSS) of 1459 in SYN and SYN/ACK packets.  We did not identify what causes this behaviour.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {C.NewM@1460TW} |
| SunOS 5.5, 5.5.1 | Y | 255 | 7(MSS) | S++ | | M@1459 | | {M@1459} |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460S} |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460STW} |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460T@0W} |
| SunOS 5.5, 5.5.1 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460TW} |
| SunOS 5.6 [64] | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {C.NewM@1460TW} |
| SunOS 5.6 | Y | 255 | 65535 | S++ | | NNTNW@1M@1460 | | {C.NewM@1460TW} |
| SunOS 5.6 | Y | 255 | 44(MSS) | S++ | | M@1459 | | {M@1459} |
| SunOS 5.6 | Y | 255 | 7(MSS) | S++ | | M@1459 | | {M@1459} |
| SunOS 5.6 | Y | 255 | 44(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.6 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.6 | Y | 255 | 44(MSS) | S++ | | M@1460 | | {M@1460S} |
| SunOS 5.6 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460S} |
| SunOS 5.6 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460STW} |
| SunOS 5.6 | Y | 255 | 65535 | S++ | | NNTNW@1M@1460 | | {M@1460STW} |
| SunOS 5.6 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460T@0W} |
| SunOS 5.6 | Y | 255 | 65535 | S++ | | NNTNW@1M@1460 | | {M@1460T@0W} |
| SunOS 5.6 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460TW} |
| SunOS 5.6 | Y | 255 | 65535 | S++ | | NNTNW@1M@1460 | | {M@1460TW} |
| SunOS 5.7 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {C.NewM@1460TW} |
| SunOS 5.7 | Y | 255 | 7(MSS) | S++ | | M@1459 | | {M@1459} |
| SunOS 5.7 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.7 | Y | 255 | 6(MSS) | S++ | | M@1460 | | {M@1460S} |
| SunOS 5.7 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460STW} |

[64] Examination of traffic traces indicates that the WIN value in SYN/ACK packet produced by SunOS 5.6 may be influenced by the network service running, the WIN value advertized in the SYN and the presence of the TCP Window scale option in the SYN packet.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| SunOS 5.7 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460T@0W} |
| SunOS 5.7 | Y | 255 | 10136 | S++ | | NNTNW@0M@1460 | | {M@1460TW} |
| SunOS 5.8 | Y | 60 [65] | 24616 | S++ | | NNTNW@0M@1460 | | {C.NewM@1460TW} |
| SunOS 5.8 | Y | 64 | 24616 | S++ | | NNTNW@0M@1460 | | {C.NewM@1460TW} |
| SunOS 5.8 | Y | 60 | 17(MSSReq)[66] | S++ | | M@1460 | | {M@1459} |
| SunOS 5.8 | Y | 64 | 17(MSSReq) | S++ | | M@1460 | | {M@1459} |
| SunOS 5.8 | Y | 60 | 17(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.8 | Y | 64 | 17(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.8 | Y | 60 | 17(MSS) | S++ | | NNSM@1460 | | {M@1460S} |
| SunOS 5.8 | Y | 64 | 17(MSS) | S++ | | NNSM@1460 | | {M@1460S} |
| SunOS 5.8 | Y | 60 | 24616 | S++ | | NNTNW@0NNSM@1460 | | {M@1460STW} |
| SunOS 5.8 | Y | 64 | 24616 | S++ | | NNTNW@0NNSM@1460 | | {M@1460STW} |
| SunOS 5.8 | Y | 60 | 24616 | S++ | | NNTNW@0M@1460 | | {M@1460T@0W} |
| SunOS 5.8 | Y | 64 | 24616 | S++ | | NNTNW@0M@1460 | | {M@1460T@0W} |
| SunOS 5.8 | Y | 60 | 24616 | S++ | | NNTNW@0M@1460 | | {M@1460TW} |
| SunOS 5.8 | Y | 64 | 24616 | S++ | | NNTNW@0M@1460 | | {M@1460TW} |
| SunOS 5.9 | Y | 60 | 49232 | S++ | | NNTM@1460NW@0 | | {C.NewM@1460TW} |
| SunOS 5.9 | Y | 64 | 49232 | S++ | | NNTM@1460NW@0 | | {C.NewM@1460TW} |
| SunOS 5.9 | Y | 60 | 34(MSSReq) | S++ | | M@1460 | | {M@1459} |
| SunOS 5.9 | Y | 64 | 34(MSSReq) | S++ | | M@1460 | | {M@1459} |
| SunOS 5.9 | Y | 60 | 34(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.9 | Y | 64 | 34(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS 5.9 | Y | 60 | 34(MSS) | S++ | | M@1460NNS | | {M@1460S} |

[65] Examination of the traffic traces indicates that the TTL value in SYN/ACK packet of SunOS 5.8 and 5.9 seems to depend on the network service running.

[66] “MSSReq” means that the TCP Window Size (WIN) in the response is related to TCP Maximum Segment Size (MSS) advertized in the request (i.e. M@1459 of the SYN packet). It can be infer from this signature that when the WIN value in a SYN/ACK transmitted by SunOS 5.8 or 5.9 is related to a MSS value, the influence comes from the MSS advertized in the SYN rather than from the MSS value advertized in the SYN/ACK response.

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| SunOS 5.9 | Y | 64 | 34(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| SunOS 5.9 | Y | 60 | 49232 | S++ | | NNTM@1460NW@0NNS | | {M@1460STW} |
| SunOS 5.9 | Y | 64 | 49232 | S++ | | NNTM@1460NW@0NNS | | {M@1460STW} |
| SunOS 5.9 | Y | 60 | 49232 | S++ | | NNTM@1460NW@0 | | {M@1460T@0W} |
| SunOS 5.9 | Y | 64 | 49232 | S++ | | NNTM@1460NW@0 | | {M@1460T@0W} |
| SunOS 5.9 | Y | 60 | 49232 | S++ | | NNTM@1460NW@0 | | {M@1460TW} |
| SunOS 5.9 | Y | 64 | 49232 | S++ | | NNTM@1460NW@0 | | {M@1460TW} |
| SunOS (Intel) 5.8 | Y | 60 | 33304 | S++ | | NNTNW@1M@1460 | | {C.NewM@1460TW} |
| SunOS (Intel) 5.8 | Y | 64 | 33304 | S++ | | NNTNW@1M@1460 | | {C.NewM@1460TW} |
| SunOS (Intel) 5.8 | Y | 60 | 44(MSSReq) | S++ | | M@1460 | | {M@1459} |
| SunOS (Intel) 5.8 | Y | 64 | 44(MSSReq) | S++ | | M@1460 | | {M@1459} |
| SunOS (Intel) 5.8 | Y | 60 | 44(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS (Intel) 5.8 | Y | 64 | 44(MSS) | S++ | | M@1460 | | {M@1460} |
| SunOS (Intel) 5.8 | Y | 60 | 44(MSS) | S++ | | NNSM@1460 | | {M@1460S} |
| SunOS (Intel) 5.8 | Y | 64 | 44(MSS) | S++ | | NNSM@1460 | | {M@1460S} |
| SunOS (Intel) 5.8 | Y | 60 | 33304 | S++ | | NNTNW@1NNSM@1460 | | {M@1460STW} |
| SunOS (Intel) 5.8 | Y | 64 | 33304 | S++ | | NNTNW@1NNSM@1460 | | {M@1460STW} |
| SunOS (Intel) 5.8 | Y | 60 | 33304 | S++ | | NNTNW@1M@1460 | | {M@1460T@0W} |
| SunOS (Intel) 5.8 | Y | 64 | 33304 | S++ | | NNTNW@1M@1460 | | {M@1460T@0W} |
| SunOS (Intel) 5.8 | Y | 60 | 33304 | S++ | | NNTNW@1M@1460 | | {M@1460TW} |
| SunOS (Intel) 5.8 | Y | 64 | 33304 | S++ | | NNTNW@1M@1460 | | {M@1460TW} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {C.NewM@1460TW} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {M@1460S} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {M@1460STW} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {M@1460T@0W} |
| Windows 95, NT 3.51 | Y | 32 | 6(MSS) | S++ | | M@1460 | | {M@1460TW} |

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {C.NewM@1460TW} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460S} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460STW} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460T@0W} |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460TW} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460 | | {C.NewM@1460TW} |
| Windows 98, 98 SE | Y | 128 | 6(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460NNS | | {M@1460STW} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460T@0W} |
| Windows 98, 98 SE | Y | 128 | 6(MSS) | S++ | | M@1460 | | {M@1460TW} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {C.NewM@1460TW} |
| Windows Millennium standard | Y | 128 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0NNS | | {M@1460STW} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460T@0W} |
| Windows Millennium standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460TW} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {C.NewM@1460TW} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0NNS | | {M@1460STW} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460T@0W} |
| Windows 2000 standard, sp2, sp3, sp4 | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460TW} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {C.NewM@1460TW} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |

| PassiveTest_TCP_SYNACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ResultOSKey | DF | TTL | WIN | AckNb | TCPecn | TCPopts | SYN_TCPecn | SYN_SetOfTCPopts |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0NNS | | {M@1460STW} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460T@0W} |
| Windows Net standard, XP Home, XP Professional | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460TW} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {C.NewM@1460TW} |
| Windows 2003 Server standard | Y | 128 | 12(MSSReq) | S++ | | M@1460 | | {M@1459} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460 | | {M@1460} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460NNS | | {M@1460S} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0NNS | | {M@1460STW} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460T@0W} |
| Windows 2003 Server standard | Y | 128 | 12(MSS) | S++ | | M@1460NW@0NNT@0 | | {M@1460TW} |

*Table 21. PassiveTest_TCP_RSTACK*

| PassiveTest_TCP_RSTACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **OS** | **DF** | **TTL** | **WIN** | **AckNb** | **TCPecn** | **Flag** | **TCPopts** | **SYN TCPecn** |
| BEOS 5 | N | 255 | 0 | S++ | | AR | | |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | 64 | 0 | S++ | | AR | | |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | N | 64 | 0 | S++ | | AR | | |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | N | 64 | 0 | S++ | | AR | | |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 255 | 0 | S++ | | AR | | |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | N | 255 | 0 | S++ | | AR | | |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | N | 255 | 0 | S++ | | AR | | |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 255 | 0 | S++ | | AR | | |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | Y | 255 | 0 | S++ | | AR | | |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB | Y | 255 | 0 | S++ | | AR | | |
| Linux 2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | 64 | 0 | S++ | | AR | | |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1 | Y | echoed[67] | 0 | S++ | | AR | | |
| MacOS 9.0 | echoed[68] | 255 | 0 | S++ | | AR | | |
| MacOS 9.1, 9.2.1, 9.2.2 | Y | 255 | 0 | S++ | | AR | | |
| MacOS 10 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5 | N | 64 | 0 | S++ | | AR | | |
| MacOS 10 10.2.6 | echoed[69] | 64 | 0 | S++ | | AR | | |

[67] MacOS 7 to 8 and SunOS 5.5 to 5.7 echo the TTL value from the SYN packet.

[68] Mac OS 9.0 appears to respond differently from the other Mac OS 9.x versions. Mac OS 9.0 echoes the IP DF bit setting of the SYN. Mac OS 9.1, 9.2.1 and 9.2.2 do not echo the DF bit.

| PassiveTest_TCP_RSTACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | WIN | AckNb | TCPecn | Flag | TCPopts | SYN TCPecn |
| NetBSD 1.1 | N | 64 | 0 | S++ | | AR | | |
| NetBSD 1.2, 1.2.1 | N | 64 | 0 | S++ | | AR | | |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | N | 64 | 0 | S++ | | AR | | |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | N | 64 | 0 | S++ | | AR | | |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 64 | 0 | S++ | | AR | | |
| NetBSD 1.6, 1.6.1 | N | 64 | 0 | S++ | | AR | | |
| Netware 4.11 | N | 128 | 0 | S++ | | AR | | |
| Netware 4.11 sp9 | Y | 128 | 0 | S++ | | AR | | |
| Netware 5, 5 sp6a | Y | 128 | 0 | S++ | | AR | | |
| Netware 5.1, 5.1 sp6 | Y | 128 | 0 | S++ | | AR | | |
| Netware 6, 6 sp3 | Y | 128 | 0 | S++ | | AR | | |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 | N | 64 | 0 | S++ | | AR | | |
| OpenBSD 2.9 | Y | 64 | 0 | S++ | | AR | | |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | Y | 64 | 0 | S++ | | AR | | |
| QNX RTP 4, 6.0 | N | 64 | echoed | S++ | | AR | Echoed | |
| QNX RTP 6.1, 6.2, 6.2.1 | N | 64 | 0 | S++ | | AR | | |
| SunOS 5.5, 5.5, 5.6, 5.7 | Y | echoed[70] | 0 | S++ | | AR | | |
| SunOS 5.8 | Y | 64 | 0 | S++ | | AR | | |
| SunOS 5.9 | Y | 64 | 0 | S++ | | AR | | |
| SunOS (Intel) 5.8 | Y | 64 | 0 | S++ | | AR | | |
| Windows 95 | N | 32 | 0 | S++ | | AR | | |
| Windows NT 3.51 standard | N | 32 | 0 | S++ | | AR | | |

[69] Examination of the traffic traces reveals that Mac OS 10.2.6 echoed the IP DF bit setting of the SYN. Mac OS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5 did not echo the DF bit.

[70] MacOS 7 to 8 and SunOS 5.5 to 5.7 echo the TTL value from the SYN packet.

| PassiveTest_TCP_RSTACK | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | WIN | AckNb | TCPecn | Flag | TCPopts | SYN TCPecn |
| Windows 98, 98 SE | N | 128 | 0 | S++ | | AR | | |
| Windows NT 4 standard, sp3, sp4, sp6 | N | 128 | 0 | S++ | | AR | | |
| Windows Millennium standard | N | 128 | 0 | S++ | | AR | | |
| Windows 2000 standard, sp2, sp3, sp4 | N | 128 | 0 | S++ | | AR | | |
| Windows XP Home, Professional | N | 128 | 0 | S++ | | AR | | |
| Windows Net standard | N | 128 | 0 | S++ | | AR | | |
| Windows 2003 Server standard | N | 128 | 0 | S++ | | AR | | |

***Table 22. PassiveTest_ICMP_Unreach***

| PassiveTest_ICMP_Unreach | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | TOS | UDPLen | IntegIPLen | IntegIPID | IntegIPFlags | IntegIPck | IntegUDPck |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1 | echoed | 255 | 0 | 8 | Y | N | N | 0 | 0 |
| FreeBSD 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | echoed | 255 | 0 | 8 | Y | N | N | nonzero | 0 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | echoed | 255 | 0 | 8 | Y | N | N | nonzero | 0 |
| FreeBSD 4.0, 4.1 | echoed | 255 | 0 | 8 | Y | N | N | nonzero | 0 |
| FreeBSD 4.1.1, 4.2, 4.3 | echoed | 255 | 0 | 8 | Y | Y | Y | nonzero | 0 |
| FreeBSD 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | echoed | 64 | 0 | 8 | Y | Y | Y | nonzero | 0 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | 64 | 192 | all | Y | Y | Y | nonzero | Y |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | 255 | 192 | all | Y | Y | Y | nonzero | Y |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB | Y | 255 | 192 | all | Y | Y | Y | nonzero | Y |
| Linux 2.4.5. 2.4.6, 2.4.7, 2.4.8, 2.4.9 | N | 255 | 192 | all | Y | Y | Y | nonzero | Y |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB | N | 255 | 192 | all | Y | Y | Y | nonzero | Y |

| PassiveTest_ICMP_Unreach | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | TOS | UDPLen | IntegIPLen | IntegIPID | IntegIPFlags | IntegIPck | IntegUDPck |
| Linux 2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk, | N | 64 | 192 | all | Y | Y | Y | nonzero | Y |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1 | Y | 255 | 0 | 64 | Y | Y | Y | nonzero | Y |
| MacOS 9 9.0 | N | 255 | 0 | 64 | Y | Y | Y | nonzero | Y |
| MacOS 9 9.1, 9.2.1, 9.2.2 | Y | 255 | 0 | 64 | Y | Y | Y | nonzero | Y |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5 | echoed | 255 | 0 | 8 | Y | Y | Y | nonzero | 0 |
| MacOS 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | echoed | 64 | 0 | 8 | Y | Y | Y | nonzero | 0 |
| NetBSD 1.1, 1.2, 1.2.1 | echoed | 255 | 0 | 8 | + | N | N | 0 | 0 |
| NetBSD 1.3, 1.3.1, 1.3.2, 1.3.3 | echoed | 255 | 0 | 8 | Y | N | N | 0 | 0 |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | N | 255 | 0 | 8 | Y | Y | Y | nonzero | Y |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | N | 255 | 0 | 8 | Y | Y | Y | nonzero | Y |
| NetBSD 1.6, 1.6.1 | N | 255 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Netware 4.11, 4.11 sp9 | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Netware 5, 5 sp6a | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Netware 5.1, 5.1 sp6 | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Netware 6, 6 sp3 | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| OpenBSD 2.0, 2.1 | echoed | 255 | 0 | 8 | + | N | N | 0 | 0 |

| PassiveTest_ICMP_Unreach | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | TOS | UDPLen | IntegIPLen | IntegIPID | IntegIPFlags | IntegIPck | IntegUDPck |
| OpenBSD  2.2, 2.3, 2.4 | echoed | 255 | 0 | 8 | Y | N | N | 0 | 0 |
| OpenBSD  2.5 | N | 255 | 0 | 8 | Y | Y | Y | nonzero | Y |
| OpenBSD  2.6, 2.7, 2.8, 2.9 | N | 255 | 0 | 8 | - | Y | Y | nonzero | Y |
| OpenBSD  3.0, 3.1, 3.2, 3.3 | N | 255 | 0 | 8 | - | Y | Y | nonzero | Y |
| QNX RTP 6.1 | N | 255 | 0 | 8 | Y | Y | Y | 0 | 0 |
| QNX RTP 6.2, 6.2.1 | N | 255 | 0 | 8 | Y | Y | Y | nonzero | Y |
| SunOS  5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | Y | 255 | 0 | 64 | Y | Y | Y | nonzero | Y |
| SunOS (Intel) 5.8 | Y | 255 | 0 | 64 | Y | Y | Y | nonzero | Y |
| Windows 95 | N | 32 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows NT 3.51 standard | N | 32 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows 98, 98 SE | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows NT 4 standard, sp3, sp4, sp6 | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows Millennium standard | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows 2000 standard, sp2, sp3, sp4 | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |
| Windows XP Home, Professional | N | 128 | 0 | 8 | Y | Y | Y | nonzero | Y |

| PassiveTest_ICMP_Unreach | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| OS | DF | TTL | TOS | UDPLen | IntegIPLen | IntegIPID | IntegIPFlags | IntegIPck | IntegUDPck |
| Windows Net standard | N | 128 | 0 | all | Y | Y | Y | nonzero | Y |
| Windows 2003 Server standard | N | 128 | 0 | all | Y | Y | Y | nonzero | Y |

***Table 23. PassiveTest_ICMP_Echo***

| PassiveTest_ICMP_Echo | | | | | | |
|---|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID | ICMPCode |
| BEOS 5 | Y | N | 255 | 0 [71] | nonzero | echoed |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | Y | echoed | 255 | echoed | nonzero | echoed |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | Y | echoed | 255 | echoed | nonzero | echoed |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3 | Y | echoed | 255 | echoed | nonzero | echoed |
| FreeBSD 4.4 , 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | Y | echoed | 64 | echoed | nonzero | echoed |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | Y | N | 64 | echoed | nonzero | echoed |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | Y | N | 255 | echoed | nonzero | echoed |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | Y | N | 255 | echoed | nonzero | echoed |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | N | 255 | echoed | nonzero | echoed |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB | Y | Y | 255 | echoed | 0 | echoed |
| Linux 2.4.5. 2.4.6, 2.4.7, 2.4.8, 2.4.9 | Y | N | 255 | echoed | nonzero | echoed |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB | Y | N | 255 | echoed | nonzero | echoed |
| Linux 2.4.18-14 | Y | N | 64 | echoed | nonzero | echoed |
| Linux 2.4.19, 2.4.19-4GB | Y | N | 64 | echoed | nonzero | echoed |
| Linux 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | N | 64 | echoed | nonzero | echoed |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | Y | Y | 255 | echoed | nonzero | echoed |
| MacOS 8.0, 8.1 | Y | Y | 255 | echoed | nonzero | echoed |
| MacOS 9.0 | Y | echoed [72] | 255 | echoed | nonzero | echoed |
| MacOS 9.1, 9.2.1, 9.2.2 | Y | Y | 255 | echoed | nonzero | echoed |
| MacOS 10.0.0, 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5 | Y | echoed | 255 | echoed | nonzero | echoed |
| MacOS 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | echoed | 64 | echoed | nonzero | echoed |
| NetBSD 1.1 | Y | echoed | 255 | echoed | nonzero | echoed |
| NetBSD 1.2, 1.2.1 | Y | echoed | 255 | echoed | nonzero | echoed |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | Y | echoed | 255 | echoed | nonzero | echoed |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | Y | echoed | 255 | echoed | nonzero | echoed |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | Y | echoed | 255 | echoed | nonzero | echoed |
| NetBSD 1.6, 1.6.1 | Y | N [73] | 255 | echoed | nonzero | echoed |
| Netware 4.11, 4.11 sp9 | Y | N | 128 | echoed | nonzero | echoed |
| Netware 5, 5 sp6a | Y | N | 128 | echoed | nonzero | echoed |
| Netware 5.1 | Y | N | 128 | 0 | nonzero | echoed |
| Netware 5.1 sp6 , 6, 6 sp3 | Y | N | 128 | echoed | nonzero | echoed |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | Y | echoed | 255 | echoed | nonzero | echoed |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | Y | echoed | 255 | echoed | nonzero | echoed |
| QNX RTP 4 | N | N/A | -1 | -1 | N/A | N/A |

[71] BeOS, Netware 5.1, and recent Windows systems do not echo the IP TOS from the stimulus.

[72] MacOS 9.0 echoes the IP DF bit setting while other Mac OS 9 set this bit to 1 independently from the stimulus.

[73] NetBSD 1.6 and 1.6.1 set the IP DF bit to 0 while other NetBSD echo the setting of the stimulus.

| PassiveTest_ICMP_Echo | | | | | | |
|---|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID | ICMPCode |
| QNX RTP 6.0 | N | N/A | -1 | -1 | N/A | N/A |
| QNX RTP 6.1, 6.2 | Y | echoed | 255 | echoed | nonzero | echoed |
| QNX RTP 6.2.1 | Y | N | 255 | echoed | nonzero | echoed |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | Y | Y | 255 | echoed | nonzero | echoed |
| SunOS (Intel) 5.8 | Y | Y | 255 | echoed | nonzero | echoed |
| Windows 95 | Y | echoed | 32 | echoed | nonzero | 0 |
| Windows NT 3.51 standard | Y | echoed | 32 | echoed | nonzero | 0 |
| Windows 98, 98 SE | Y | echoed | 128 | echoed | nonzero | 0 |
| Windows NT 4 standard, sp3, sp4, sp6 | Y | echoed | 128 | echoed | nonzero | 0 |
| Windows Millennium standard | Y | echoed | 128 | echoed | nonzero | 0 |
| Windows 2000 standard, sp2, sp3, sp4 | Y | echoed | 128 | 0[74] | nonzero | 0 |
| Windows XP Home, Professional | Y | echoed | 128 | 0 | nonzero | 0 |
| Windows Net standard | Y | echoed | 128 | 0 | nonzero | 0 |
| Windows 2003 Server standard | Y | echoed | 128 | 0 | nonzero | 0 |

***Table 24. PassiveTest_ICMP_Info***

| PassiveTest_ICMP_Info | | | | | |
|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID |
| BEOS 5 | N | N/A | -1 | -1 | N/A |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | N/A | -1 | -1 | N/A |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | N | N/A | -1 | -1 | N/A |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | N | N/A | -1 | -1 | N/A |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | N/A | -1 | -1 | N/A |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | N | N/A | -1 | -1 | N/A |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | N | N/A | -1 | -1 | N/A |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | N/A | -1 | -1 | N/A |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | N | N/A | -1 | -1 | N/A |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | N/A | -1 | -1 | N/A |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | N | N/A | -1 | -1 | N/A |
| MacOS 8.0, 8.1 | N | N/A | -1 | -1 | N/A |
| MacOS 9.0, 9.1, 9.2.1, 9.2.2 | N | N/A | -1 | -1 | N/A |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.1 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.2, 1.2.1 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | N | N/A | -1 | -1 | N/A |
| NetBSD 1.6, 1.6.1 | N | N/A | -1 | -1 | N/A |
| Netware 4.11, 4.11 sp9 | N | N/A | -1 | -1 | N/A |
| Netware 5, 5 sp6a, | N | N/A | -1 | -1 | N/A |

[74] BeOS, Netware 5.1, and recent Windows systems do not echo the IP TOS from the stimulus.

| PassiveTest_ICMP_Info | | | | | |
|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID |
| Netware 5.1, 5.1 sp6 | N | N/A | -1 | -1 | N/A |
| Netware 6 , 6 sp3 | N | N/A | -1 | -1 | N/A |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | N | N/A | -1 | -1 | N/A |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | N | N/A | -1 | -1 | N/A |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | N | N/A | -1 | -1 | N/A |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | N | N/A | -1 | -1 | N/A |
| SunOS (Intel) 5.8 | N | N/A | -1 | -1 | N/A |
| Windows 95 | N | N/A | -1 | -1 | N/A |
| Windows NT 3.51 standard | N | N/A | -1 | -1 | N/A |
| Windows 98, 98 SE | N | N/A | -1 | -1 | N/A |
| Windows NT 4 standard, sp3, sp4, sp6 | N | N/A | -1 | -1 | N/A |
| Windows Millennium standard | N | N/A | -1 | -1 | N/A |
| Windows 2000 standard, sp2, sp3, sp4 | N | N/A | -1 | -1 | N/A |
| Windows XP Home, Professional | N | N/A | -1 | -1 | N/A |
| Windows Net standard | N | N/A | -1 | -1 | N/A |
| Windows 2003 Server standard | N | N/A | -1 | -1 | N/A |

***Table 25. PassiveTest_ICMP_TS***

| PassiveTest_ICMP_TS | | | | | |
|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID |
| BEOS 5 | N | N/A | -1 | N/A | N/A |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | Y | echoed | 255 | echoed | nonzero |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | Y | echoed | 255 | echoed | nonzero |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3 | Y | echoed | 255 | echoed | nonzero |
| FreeBSD 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8 | Y | echoed | 64 | echoed | nonzero |
| FreeBSD 5.0, 5.1 | Y | echoed | 64 | echoed | nonzero |
| Linux 2.0.30, 2.32, 2.0.36 (all Red Hat) | N | N/A | -1 | N/A | N/A |
| Linux 2.0.29, 2.0.34, 2.0.36 (all Debian) | Y | N | 64 | echoed | nonzero |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | Y | N | 255 | echoed | nonzero |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | Y | N | 255 | echoed | nonzero |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | Y | N | 255 | echoed | nonzero |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB | Y | Y | 255 | echoed | 0 |
| Linux 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | Y | N | 255 | echoed | nonzero |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB | Y | N | 255 | echoed | nonzero |
| Linux 2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | Y | N | 64 | echoed | nonzero |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | N | N/A | -1 | N/A | N/A |
| MacOS 8.0, 8.1 | N | N/A | -1 | N/A | N/A |
| MacOS 9.0 | N | N/A | -1 | N/A | N/A |
| MacOS 9.1, 9.2.1, 9.2.2 | N | N/A | -1 | N/A | N/A |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5 | Y | echoed | 255 | echoed | nonzero |
| MacOS 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | Y | echoed | 64 | echoed | nonzero |
| NetBSD 1.1 | Y | echoed | 255 | echoed | nonzero |
| NetBSD 1.2, 1.2.1 | Y | echoed | 255 | echoed | nonzero |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | Y | echoed | 255 | echoed | nonzero |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | Y | echoed | 255 | echoed | nonzero |

| PassiveTest_ICMP_TS | | | | | |
|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | Y | echoed | 255 | echoed | nonzero |
| NetBSD 1.6, 1.6.1 | Y | N [75] | 255 | echoed | nonzero |
| Netware 4.11, 4.11 sp9 | N | N/A | -1 | N/A | N/A |
| Netware 5, 5 sp6a | N | N/A | -1 | N/A | N/A |
| Netware 5.1, 5.1 sp6 | N | N/A | -1 | N/A | N/A |
| Netware 6, 6 sp3 | N | N/A | -1 | N/A | N/A |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | Y | echoed | 255 | echoed | nonzero |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | Y | echoed | 255 | echoed | nonzero |
| QNX RTP 4 | N | N/A | -1 | N/A | N/A |
| QNX RTP 6.0 | N | N/A | -1 | N/A | N/A |
| QNX RTP 6.1, 6.2 | Y | echoed | 255 | echoed | nonzero |
| QNX RTP 6.2.1 | Y | N | 255 | echoed | nonzero |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | Y | Y | 255 | echoed | nonzero |
| SunOS (Intel) 5.8 | Y | Y | 255 | echoed | nonzero |
| Windows 95 | N | N/A | -1 | N/A | N/A |
| Windows NT 3.51 standard | N | N/A | -1 | N/A | N/A |
| Windows 98, 98 SE | Y | N | 128 | 0 [76] | nonzero |
| Windows NT 4 standard, sp3, sp4, sp6 | N | N/A | -1 | N/A | N/A |
| Windows Millennium standard | Y | N | 128 | 0 | nonzero |
| Windows 2000 standard, sp2, sp3, sp4 | Y | N | 128 | 0 | nonzero |
| Windows XP Home, Professional | Y | N | 128 | 0 | nonzero |
| Windows Net standard | Y | N | 128 | 0 | nonzero |
| Windows 2003 Server standard | Y | N | 128 | 0 | nonzero |

***Table 26. PassiveTest_ICMP_Mask***

| **PassiveTest_ICMP_Mask** | | | | | |
|---|---|---|---|---|---|
| **OS** | **Resp** | **DF** | **TTL** | **TOS** | **IPID** |
| BEOS 5 | N | N/A | N/A | N/A | N/A |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | N | N/A | N/A | N/A | N/A |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | N | N/A | N/A | N/A | N/A |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 5.0, 5.1 | N | N/A | N/A | N/A | N/A |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | N | N/A | N/A | N/A | N/A |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | N | N/A | N/A | N/A | N/A |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | N | N/A | N/A | N/A | N/A |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | N | N/A | N/A | N/A | N/A |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | N | N/A | N/A | N/A | N/A |

[75] NetBSD 1.6 and 1.6.1 set the DF bit to 0 while other NetBSD echo the setting of the stimulus.

[76] Windows systems do not echo the IP TOS from the stimulus.

| PassiveTest_ICMP_Mask | | | | | |
|---|---|---|---|---|---|
| OS | Resp | DF | TTL | TOS | IPID |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB, 2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | N | N/A | N/A | N/A | N/A |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1 | Y | Y | 255 | echoed | nonzero |
| MacOS 8.0, 8.1 | Y | Y | 255 | echoed | nonzero |
| MacOS 9.0 | Y | echoed [77] | 255 | echoed | nonzero |
| MacOS 9.1, 9.2.1, 9.2.2 | Y | Y | 255 | echoed | nonzero |
| MacOS 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.1 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.2, 1.2.1 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.3 , 1.3.1, 1.3.2, 1.3.3 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.4 , 1.4.1, 1.4.2, 1.4.3 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.5, 1.5.1, 1.5.2, 1.5.3 | N | N/A | N/A | N/A | N/A |
| NetBSD 1.6, 1.6.1 | N | N/A | N/A | N/A | N/A |
| Netware 4.11, 4.11 sp9 | N | N/A | N/A | N/A | N/A |
| Netware 5, 5 sp6a | Y | N | echoed | echoed | nonzero |
| Netware 5.1 [78] | Y | N | echoed | 0 | nonzero |
| Netware 5.1 sp6 | N | N/A | N/A | N/A | N/A |
| Netware 6 | Y | N | echoed | echoed | nonzero |
| Netware 6 sp3 | N | N/A | N/A | N/A | N/A |
| OpenBSD 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9 | N | N/A | N/A | N/A | N/A |
| OpenBSD 3.0, 3.1, 3.2, 3.3 | N | N/A | N/A | N/A | N/A |
| QNX RTP 4 | N | N/A | N/A | N/A | N/A |
| QNX RTP 6.0, 6.1, 6.2, 6.2.1 | N | N/A | N/A | N/A | N/A |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | Y | Y | 255 | echoed | nonzero |
| SunOS (Intel) 5.8 | Y | Y | 255 | echoed | nonzero |
| Windows 95 | Y | N | 32 | 0 [79] | nonzero |
| Windows NT 3.51 standard | Y | N | 32 | 0 | nonzero |
| Windows 98, 98 SE | Y | N | 128 | 0 | nonzero |
| Windows NT 4 standard, sp3 | Y | N | 128 | 0 | nonzero |
| Windows NT 4 sp4, sp6 | N | N/A | N/A | N/A | N/A |
| Windows Millennium standard | N | N/A | N/A | N/A | N/A |
| Windows 2000 standard, sp2, sp3, sp4 | N | N/A | N/A | N/A | N/A |
| Windows XP Home, Professional | N | N/A | N/A | N/A | N/A |
| Windows Net standard | N | N/A | N/A | N/A | N/A |
| Windows 2003 Server standard | N | N/A | N/A | N/A | N/A |

***Table 27. PassiveTest_ICMP_ID_SEQ***

| PassiveTest_ICMP_ID_SEQ | | | | |
|---|---|---|---|---|
| OS | ICMPIDClass | IDinvariant | ICMPSeqClass | SeqInvariant |

[77] MacOS 9.0 echoes the DF bit while other Mac OS 9 do not.

[78] Service packs of Novell Netware 5.1 and 6 prevent the sytem from giving away mask information.

[79] Windows systems do not echo the IP TOS from the stimulus.

PassiveTest_ICMP_ID_SEQ

| OS | ICMPIDClass | IDInvariant | ICMPSeqClass | SeqInvariant |
|---|---|---|---|---|
| BEOS 5 | I | 1 | I | 100 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | TDI (or I) | 100 | I | 100 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | TDI (or I) | 100 | I | 100 |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4 , 4.5, 4.6, 4.6.2, 4.7, 4.8, | TDI (or I) | 100 | I | 100 |
| FreeBSD 5.0, 5.1 | TDI (or I) | 100 | I | 1 [80] |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | TDI (or I) | 100 | I | 100 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | TDI (or I) | 100 | I | 100 |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19 | TDI (or I) | 100 | I | 100 |
| Linux 2.2.16 (S.u.S.E 7.0), Linux 2.2.18 (S.u.S.E 7.1) | TDI (or I) | 100 | I | 1 [81] |
| Linux 2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | TDI (or I) | 100 | I | 100 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | TDI (or I) | 100 | I | 100 |
| Linux 2.4.4-4GB (S.u.S.E 7.2) | TDI (or I) | 100 | I | 1 [82] |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB,2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20 | TDI (or I) | 100 | I | 100 |
| Linux 2.4.20-8 (RedHat 9) [83] | TDI (or I) | 100 | I | 1 |
| Linux 2.4.21-0.13mdk (Mandrake PPC 9.1) | TDI (or I) | 1 | I | 1 |
| MacOS 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.0, 9.1, 9.2.2 (using TCPMac Ping) | I | 1 | C | 0 |
| MacOS 10.0.0, 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | TDI (or I) | 1 | I | 1 |
| NetBSD 1.1, 1.2, 1.2.1 [84] | TDI (or I) | 100 | I | 100 |
| NetBSD 1.3, 1.3.1, 1.3.2, 1.3.3 | TDI (or I) | 100 | I | 1 |
| NetBSD 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | TDI (or I) | 1 | I | 1 |
| Netware 4.11, 4.11 sp9 | I | 6000 | I | 100 |
| OpenBSD 2.0, 2.1 | TDI (or I) | 100 | I | 100 |
| OpenBSD 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | RD | -1 | I | 1 |
| QNX RTP 4, 6.0, 6.1, 6.2 | TDI (or I) | 10 | I | 1 |
| QNX RTP 6.2.1 | I | 2000 | I | 1 |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | TDI (or I) | 1 | I | 1 |
| SunOS (Intel) 5.8 | TDI (or I) | 1 | I | 1 |
| Windows 95 | C | 100 | IGlobal | 100 |
| Windows NT 3.51 standard | C | 100 | IGlobal | 100 |
| Windows 98, 98 SE | C | 200 | IGlobal | 100 |
| Windows NT 4 standard, sp3, sp4, sp6 | C | 100 | IGlobal | 100 |
| Windows Millennium standard | C | 300 | IGlobal | 100 |
| Windows 2000 standard, sp2, sp3, sp4 | C | 200 | IGlobal | 100 |
| Windows XP Home, Professional | C | 200 | IGlobal | 100 |
| Windows Net standard | C | 200 | IGlobal | 100 |
| Windows 2003 Server standard | C | 200 | IGlobal | 100 |

[80] FreeBSD 5.0 and 5.1 increments the ICMP Sequence number by 0x0001 while older FreeBSD increment it by 0x0100.

[81] Linux 2.2.16 and 2.2.18 distributed by S.u.S.E have a signature different than other 2.2.16 and 2.2.18 kernels.

[82] Linux 2.4.4-4GB (S.u.S.E.) has a different signature than other 2.4.4 kernels.

[83] The RedHat 9 and Mandrake PPC 9.1 distributions have their own distinct signatures.

[84] There are three different signatures to distinguish between the NetBSD versions.

*Table 28. PassiveTest_ICMP_ID (Subtest of PassiveTest_ICMP_ID_SEQ)*

| PassiveTest_ICMP_ID (Subtest of PassiveTest_ICMP_ID_SEQ) | | | | | |
|---|---|---|---|---|---|
| ResultOSKey | ICMPID | DF | TOS | DataLen | ConstantData |
| BEOS 5 | other | N | 0 | 30 | 0000000000000000000008090A0B0C0D0E0F10111213 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4 , 4.5, 4.6, 4.6.2, 4.7, 4.8 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| FreeBSD 5.0 , 5.1 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19,2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.4-4GB, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | other | Y | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB,2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20, 2.4.20-8, 2.4.21-0.13mdk | other | Y | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| MacOS 7.5.3, 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1 (using TCPMac Ping) | other | Y | 0 | 56 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |
| MacOS 9 9.0 (using TCPMac Ping) | other | N[85] | 0 | 56 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |

[85] MacOS 9.0 appears to have a different signature than MacOS 9.1 and 9.2.  This requires further investigation.

| PassiveTest_ICMP_ID (Subtest of PassiveTest_ICMP_ID_SEQ) | | | | | |
|---|---|---|---|---|---|
| ResultOSKey | ICMPID | DF | TOS | DataLen | ConstantData |
| MacOS 9 9.1, 9.2.1, 9.2.2 (using TCPMac Ping) | other | Y | 0 | 56 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |
| MacOS 10.0.0, 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| NetBSD 1.1, 1.2, 1.2.1, 1.3, 1.3.1, 1.3.2, 1.3.3 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| NetBSD 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Netware 4.11, 4.11 sp9 | other | N | 0 | 12 | 0000 |
| Netware 5, 5 sp6a, 5.1, 5.1 sp6 | other | N | 0 | 12 | 0000 |
| Netware 6, 6 sp3 | other | N | 0 | 12 | 0000 |
| OpenBSD 2.0, 2.1, 2.2, 2.3 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| OpenBSD 2.4 | other | N | 0 | 56 | 101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F30313233343536370000000000000000[86] |
| OpenBSD 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | other | N | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | other | Y | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| SunOS (Intel) 5.8 | other | Y | 0 | 56 | 08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 |
| Windows 95 | 100 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows NT 3.51 standard | 100 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows 98, 98 SE | 200 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows NT 4 standard, sp3, sp4, sp6 | 100 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows Millennium standard | 300 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows 2000 standard, sp2, sp3, sp4 | 200 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows XP Home, Professional | 200 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows Net standard | 200 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |
| Windows 2003 Server standard | 200 | N | 0 | 32 | 6162636465666768696A6B6C6D6E6F7071727374757677616263646566676869 |

[86] The data transmitted by the ping utility version installed by default on OpenBSD 2.4 begins at a 8-byte offset from the usual data string. The ping version is 1.35. The error was fixed in later releases.

*Table 29. PassiveTest_ICMP_SEQ (Subtest of PassiveTest_ICMP_ID_SEQ)*

| PassiveTest_ICMP_SEQ (Subtest of PassiveTest_ICMP_ID_SEQ) | | |
|---|---|---|
| ResultOSKey | ICMPSeqClass | SeqInvariant |
| BEOS 5 | I | 100 |
| FreeBSD 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.7.1, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8 | I | 100 |
| FreeBSD 3.0, 3.1, 3.2, 3.3, 3.4, 3.5.1 | I | 100 |
| FreeBSD 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4 , 4.5, 4.6, 4.6.2, 4.7, 4.8 | I | 100 |
| FreeBSD 5.0 , 5.1 | I | 1 |
| Linux 2.0.29, 2.0.30, 2.0.32, 2.0.34, 2.0.36 | I | 100 |
| Linux 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.5-15, 2.2.6, 2.2.7, 2.2.8, 2.2.9 | I | 100 |
| Linux 2.2.10, 2.2.11, 2.2.12, 2.2.12-20, 2.2.13, 2.2.14, 2.2.14-5, 2.2.15, 2.2.16, 2.2.16-22, 2.2.17, 2.2.18, 2.2.19,2.2.20, 2.2.20-idepci, 2.2.21, 2.2.22, 2.2.23, 2.2.24 | I | 100 |
| Linux 2.2.16 (S.u.S.E 7.0), Linux 2.2.18 (S.u.S.E 7.1) | I | 1 |
| Linux 2.4.0, 2.4.1, 2.4.2, 2.4.2-2, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.4.7, 2.4.8, 2.4.9 | I | 100 |
| Linux 2.4.4-4GB | I | 1 |
| Linux 2.4.10, 2.4.10-4GB, 2.4.11, 2.4.12, 2.4.13, 2.4.14, 2.4.15, 2.4.16, 2.4.17, 2.4.18, 2.4.18-3, 2.4.18-4GB,2.4.18-14, 2.4.19, 2.4.19-4GB, 2.4.20 | I | 100 |
| Linux 2.4.20-8 | I | 1 |
| Linux 2.4.21-0.13mdk | I | 1 |
| MacOS 10.0.0, 10.1.0, 10.1.1, 10.1.2, 10.1.3, 10.1.4, 10.1.5, 10.2.1, 10.2.2, 10.2.3, 10.2.4, 10.2.5, 10.2.6 | I | 1 |
| MacOS 7.5.3, 7.5.3, 7.5.5, 7.6, 7.6.1, 8.0, 8.1, 9.0, 9.1, 9.2.2 (using TCPMac Ping) | C | 0 |
| NetBSD 1.1, 1.2, 1.2.1 | I | 100 |
| NetBSD 1.3, 1.3.1, 1.3.2, 1.3.3 | I | 1 |
| NetBSD 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1 | I | 1 |
| Netware 4.11, 4.11 sp9 | I | 100 |
| Netware 5, 5 sp6a, 5.1, 5.1 sp6 | I | 100 |
| Netware 6, 6 sp3 | I | 100 |
| OpenBSD 2.0, 2.1 | I | 100 |
| OpenBSD 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3 | I | 1 |
| QNX RTP 4, 6.0, 6.1, 6.2, 6.2.1 | I | 1 |
| SunOS 5.5, 5.5.1, 5.6, 5.7, 5.8, 5.9 | I | 1 |
| SunOS (Intel) 5.8 | I | 1 |
| Windows 95 | I | 100 |
| Windows NT 3.51 standard | I | 100 |
| Windows 98, 98 SE | I | 100 |
| Windows NT 4 standard, sp3, sp4, sp6 | I | 100 |
| Windows Millennium standard | I | 100 |
| Windows 2000 standard, sp2, sp3, sp4 | I | 100 |
| Windows XP Home, Professional | I | 100 |
| Windows Net standard | I | 100 |
| Windows 2003 Server standard | I | 100 |

# Annex B: Active OS identification tools: Analysis of Nmap and Xprobe

This Annex describes two active OS identification tools : Nmap, and Xprobe. The descriptions are based on analysis of the programs' code and on analysis of traffic they generate. Nmap and Xprobe combine a great deal of the OS fingerprinting techniques currently known. For other interesting methods not covered by Nmap, and Xprobe, the reader can refer to [2], [3] and [11].

The Annex is broken into two sections, one for each tool. In each section, we describe in detail the packets aimed at the target and the checks conducted on the response packets. We end each section with some general remarks on OS differences that were observed during the analysis of the tool.

## OS scan with Nmap

Nmap is a network mapper utility designed to scan large networks. It is capable of determining what hosts are available on the network, what services they are offering, what operating system they are running, what type of packet filters/firewalls are in use, and several of other characteristics. We focus here on its OS detection capability. Nmap's fingerprinting technique tests the TCP/IP stack implementation of the target by sending craft packets and observing the responses. The technique is broken down into nine tests. As of November 2004, the version available for download from nmap's website is 3.75. The version examined during this study was 2.54Beta29. While the new version is still based on the nine tests described below, some discrepancies should be expected.

Nmap stores all of its known Operating System (OS) fingerprints in a text file named "nmap-os-fingerprint". A typical entry in this file is provided below.

```
Fingerprint Linux 2.1.19 - 2.2.17

TSeq(Class=RI%gcd=<8%SI=>10000%IPID=I%TS=100HZ)

T1(DF=Y|N%W=3C0A|3F25|7B2F|7F53|7C38|B63%ACK=S++%Flags=AS%Ops=MENNTNW)

T2(Resp=N)

T3(Resp=Y|N%DF=Y%W=3C0A|3F25|7B2F|7F53|7C38|B63%ACK=S++%Flags=AS%Ops=MENNTNW)

T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

T6(DF=N%W=0%ACK=O%Flags=R%Ops=)

T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)

PU(DF=N%TOS=C0|A0|0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)
```

This particular entry associates the results of the nine tests (Tseq, T1, T2, T3, T4, T5, T6, T7, PU) to the fingerprint of a machine running Linux with kernel versions 2.1.19 to 2.2.17. These tests are described in more detail in the sections below.

To proceed with the OS scan, Nmap targets two ports on the victim's machine: one in the state OPEN and one in the state CLOSED. To find these two ports, Nmap starts with a port scan on the target. The tool however, has an option that allows the user to limit the number of ports to be scanned. If Nmap finds no closed ports among the user-specified ones, it will pick a presumed closed port (greater than 30000).

## Description of tests T1-T7

Each test T1 to T4 consists in sending one TCP packet to an open port. They differ by their TCP flags and, in the T1 case, by the use of a reserved bit. This bit is the one preceding the URG flag bit and its use is not standard since it is part of a reserved field as specified by RFC 793 (the protocol standard for TCP). RFC 3168 (proposed standard) proposes that the last two bits of the reserved field be used for control of congestion. These two special bits are referred to as CWR and ECN respectively. The tests T5 through T7 send one TCP packet each to a closed port and differ only by their TCP flags.

All of these seven crafted packets are sent with the following TCP options:

1. WINDOW SCALE of 10B,

2. NOP,

3. MSS of 265B,

4. TIMESTAMP,

5. EOL.

These options appear in this very specific order. The IP datagrams have total length of 60B ( 20B of IP header + 40B of TCP header with options + 0B of data ). The following list summarises the crafted packets with their TCP flags settings.

T1: [*SYN, ECN*] packet with TCP options sent to an **open** port.

T2: NULL packet (none of the flags set) with TCP options sent to an **open** port.

T3: [*SYN, FIN, URG, PSH*] packet with urgent pointer set to 0. The packet is sent with TCP options to an **open** port.

T4: [*ACK*] packet with TCP options sent to an **open** port.

T5: [*SYN*] packet with TCP options sent to a **closed** port.

T6: [*ACK*] packet with TCP options sent to a **closed** port.

T7: [*FIN, URG, PSH*] packet with urgent pointer set to 0. The packet is sent with TCP options to a **closed** port.

Nmap captures and analyses the response of the target for each of these packets. More specifically, it will categorize the response based on six check criterions:

1. Did we receive a response to this packet? (“Y” or “N”)

2. Is the *Don’t Fragment* bit set in the target’s response? (“Y” or “N”)

3. What is the TCP *Window size* value in the target’s response? (two-byte integer expressed in hexadecimal)

4. What is the TCP *Acknowledgement number* in the target’s response (in relation to the sequence number of nmap’s triggering packet)? (“S”, or “S++”, or “O”) [87]

5. What TCP flags are included in the target’s response? (a subset of “B, U, A, P, R, S, F”) [88]

6. What TCP options are included in the target’s response packet? [89] (“L”, “N”, “M”, “E”, “W”, “T”) [90]

The fingerprint format makes use of “%” symbols to separate the responses to the above criterions. When multiple responses are possible, they are separated by a “|” symbol.

For example, consider the line beginning with T3 from the Linux example,

T3(Resp=Y|N%DF=Y%W=3C0A|3F25|7B2F|7F53|7C38|B63%ACK=S++%
Flags=AS%Ops=MENNTNW)

It says that the response packet to Nmap’s T3 test would have the following characteristics:

[87] “S” if the *Acknowledgment number* of the response is equal to the *Sequence number* of the triggering packet, “S++” if it is incremented by one, or “O” for any other value.

[88] “B” stands for the BOGUS bit because it is supposed to be unused. This bit is the one preceding the *URG* bit, that is the ECN as discussed earlier.

[89] Options sent back (and thus supported) by the targeted. They can appear in a different order depending of the target’s operating system.

[90] “L” if *End of option list* (code 0) is set,
“N” if *No-operation* option (code 1) is set,
“M” if *MSS* option (code 2) is set, followed by “E” if the value is echoed from the nmap packet’s MSS,
“W” if *Window scale* option (code 3) is set,
“T” if *Timestamp* option (code 8) is set.

- Resp=Y|N The target may or may not respond.
- DF=Y The response packet is expected to have the *Don't fragment* bit set.
- W=3C0A|3F25|7B2F|7F53|7C38|B63 The Window size value expected is one of the following: 3C0A, or 3F25, or 7B2F, or 7F53, or 7C38, or B63.
- ACK=S++ The response's *Acknowledgement number* is expected to be equal to triggering packet's *Initial sequence number* plus 1.
- Flags=AS The TCP flags expected are SYN and ACK.
- Ops=MENNTNW The response packet should have the following options set in this order: <MSS(Echoed)><NOP><NOP><Timestamp> <NOP><Window scale>

Note that the first check,"Resp=", is absent from T1, T4, T5, T6, and T7 fingerprints. This means that lack of a response will not disqualify a match as long as all the other tests of the fingerprint structure match. Nmap uses this strategy because generally, Operating Systems do respond to these packets. Therefore, a lack of response to these packets is more likely attributed to the network conditions and not the OS itself. For instance, they could be dropped by a firewall. The criteria "Resp=" appears in T2 and T3 because some operating system do drop those without responding.

## Description of the PU test

PU stands for "port unreachable". This test probes an ICMP *port unreachable* message by sending one UDP packet to a closed UDP port. The IP datagram total length of the triggering packet is 328B (20B of IP header + 8B of UDP header + 300B of data). The data consist of a certain repeated byte. The "pattern-byte" is picked randomly each time Nmap sends out this test.

<u>**Recall**</u>: The content of an IP datagram encapsulating an ICMP port unreachable error message is as follows:

| |
|---|
| **IP header** |
| **ICMP header** |
| **Data, consisting of the firsts few bytes of the packet that triggered the ICMP error message, thus in this particular case:**<br>IP header of the crafted packet<br>+<br>First “few”[91] bytes of the UDP header and data of the crafted packet |

Nmap categorises the response based on nine criteria described below. The six last are concerned by the ICMP data content, which is the echoed IP datagram that caused the ICMP packet to be sent. Nmaps looks at how much of its original packet is echoed, and also verifies if some particular fields were modified.

1. Did we receive a response (an ICMP port unreachable message) to this packet? (“Y” or “N”)

2. Is the *Don’t Fragment* bit set in the target’s response? (“Y” or “N”)

3. What Type of service (TOS) is set in the response packet? (hexadecimal value)

4. What is the value of the *IP total length* field of the response packet? (hexadecimal value)

5. What is the *IP total length* field of the **offending packet** being echoed? (hexadecimal value)

6. Has the IP *Identification* field of the **offending packet** been echoed correctly? (“0”, “E”, “F”)[92]

7. Does the IP checksum of the **offending packet** being echoed computes? (“0”, “E”, “F”)[93]

8. Does the UDP checksum of the **offending packet** being echoed computes? (“0”, “E”, “F”)

9. What is the value of the UDP *Message length* field of the **offending packet** being echoed? (hexadecimal value)

[91] At least the first 8 bytes following the IP header. More than 8 bytes may be sent according to RFC 1122 (Requirements for Internet Host – Communication Layers), section 3.2.2. RFC 1122 is an official standard.

[92] “0” if the returned value is zero, “E” (for “as Expected”) if the returned value is correctly echoed, and “F” otherwise.

[93] “0” if the returned value is zero, “E” (for “as Expected”) if the checksum computes, and “F” otherwise.

10. Have the Data of the **offending packet** been echoed correctly? (“E”, “F”)

For example, consider the line beginning with PU from the Linux example,

PU(DF=N%TOS=C0|A0|0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E
%UCK=E|F%ULEN=134%DAT=E)

It says that the response packet to Nmap’s PU test would have the following characteristics:

- Resp is absent The target normally responds to this packet. If no response is received for PU test, do not discard the fingerprint because of it.
- DF=N The *Don’t fragment* is expected to be 0 (not set).
- TOS=C0 The Type of Service expected should be set to one of these values: C0, A0, or 0.
- IPLEN=164 The IP total length expected hexadecimal value is 0x0164 (i.e. 356 bytes)
- RIPTL=148 The data corresponding to the IP total length field of the returned datagram is expected to be 0x0148 (i.e. 328 bytes)
- RID=E The data corresponding to the IP ID field of the returned datagram is expected to be correctly echoed.
- RIPCK=E The data corresponding to the IP Checksum field of the returned datagram is expected to compute correctly.
- UCK=E|F The data corresponding to the UDP Checksum field of the returned datagram may be correct or incorrect.
- ULEN=134 The data corresponding to the UDP Message length field of the returned datagram is expected to have the value 0x0134 (i.e. 308 bytes)
- DAT=E The data corresponding to the returned datagram’s data field is expected to be echoed correctly.

## Description of test Tseq

This test investigates the predictability of the TCP *Initial Sequence Numbers* (ISN) as well as the IP *Identification* (ID) numbers generation. It also tries to characterize the TCP timestamp clock update rate of the target’s operating system.

ISN numbers are generated during the first and second handshake of a TCP connection. The initiator of the connection sends his ISN in the SYN packet, and the responding host sends his in the SYN/ACK packet. New IP identification numbers are generated for every IP packets. As for the TCP timestamp clock values, they appear when the TCP timestamp option is set.

The Tseq test differs from the other Nmap tests in the sense that it cannot be based solely on one packet, it requires constructing a sample of packets sent by the target. Nmap uses a six-packet sample composed of SYN/ACK packets received from the target. To get these packets, Nmap initiates six consecutive connections[94] in a very short amount of time. If at least 4 responses are received and that the delay between the probes[95] is no longer than one second, nmap considers the sample as being suitable for its calculations.

The crafted packets that Nmap sends out for this test have the TCP SYN flag set and have a total length of 60B ( 20B of IP header + 40B of TCP header with options + 0B of data ). The options are identical to the ones of the tests T1 through T7.

For each response packet, Nmap collects the values of the following fields:

- The Initial sequence number of the TCP header
- The *Identification* field of the IP header
- The timestamp value (*tsval*) of the TCP Timestamp option (if supported).

It then looks at how these values differ from one packet to another. The following paragraphs summarise the different categories nmap defines to characterize the target behaviour:

The ISN classes defined by Nmap are:

- Constant ISNs (Class C)
- ISNs that are multiple of 64000 (Class 64K)
- ISNs that are multiple of 800 (Class i800)
- ISNs incremented using random positive increments (Class RI)
- ISNs produced by a true random generator (Class TR)
- Time dependent ISNs (Class TD)

[94] The connections are never fully opened however since Nmap tears them down by sending RST packets immediately after receiving SYN/ACK packets. That is, only the first two handshakes of the TCP three-way handshakes are completed. This is also known as “half-open scanning”.

[95] nmap has an option that allows the user to set the delay between the transmission of each stimulus. The longer the delay, the stealthier the tool is. However, in cases where the location of the system running nmap prevents it from seeing all of its target's traffic, a longer delay increases the likelihood of getting non-consecutive ISNs. This is because the target can communicate with other hosts during the sampling period.

The IP ID classes defined by Nmap are:

- IDs incremented by one each time (Class I)
- IDs incremented by 256 each time (Class BI)[96]
- IDs incremented using random positive increments (Class RPI)
- IDs coming from a random distribution (Class RD)
- Repeatable IDs (Class C)
- Zeroed out IDs (Class Z)[97]

The Timestamp classes defined by Nmap are:

- Timestamp clocks updated twice per second (Class 2HZ)
- Timestamp clocks updated 100 times per second (Class 100HZ)
- Timestamp clocks updated 1000 times per second (Class 1000HZ)
- Timestamp option not supported by the OS (Class U)
- Timestamp option set but having a value of zero (Class Z)[98]

The algorithms Nmap uses to classify the target into these different categories are rather simplistic. For instance, IP IDs sampled are said to come from a random distribution if at least one ID number is smaller then its predecessor. While this is indeed an indicator of a random distribution, a particular sample may not present this characteristics, but may still come from such a distribution. That being said, Nmap still achieves good accuracy using its algorithms.

[96] "BI" stands for "broken increment". This 256-incremental behavior is seen on some little endian platforms when the operating system "forgets" to reorder the bytes. Windows 95 falls into this "Broken incremental" IP IDs category.

[97] Linux kernel 2.4 falls into this category. This doesn't mean that this system sends all packets with an IP ID of zero, but it does do it for all of its SYN/ACK packets.

[98] Windows 2000 falls into this category. It supports the option, but waits until the tree-way handshake is completed before sending any timestamp value.

To give an example of how to interpret the TSeq test results of a fingerprint structure, let’s look at the line beginning with TSeq from the Linux example,

TSeq(Class=RI%gcd=<8%SI=>10000%IPID=I%TS=100HZ)

It says that the response packet to Nmap’s TSeq test would find the following characteristics:

- Class=RI The ISNs are randomly incremented
- Gcd=<8 The Greater Common Divisor of the ISN differences is expected to be smaller than 8 (hexadecimal value).
- SI=>10000 The standard deviation of the ISN differences is expected to be greater than 10000.
- IPID=I The IP IDs are incremented by one.
- TS=100HZ The timestamp clock is updated 100 times per second.

## General Remarks on OS differences

A quick analysis of the nmap-os-fingerprint file (from Nmap’s version 2.54Beta29) allows to draw the following observations:

- Windows systems respond to every test [99];
- Linux, OpenBSD, FreeBSD, Solaris systems don’t response to T2[100];
- The setting of the TCP options in response to the T1 test may be used to differentiate between broad families of OS. For instance,
  - Mac OS systems prior to OS X tend to respond with MEWL or MEWNNNT;
  - Solaris systems tend to respond with NNTWM or NNTNWME;
  - Linux systems tend to respond with MNNTNW (newer kernel version), or MENNTNW (older kernel version);
  - OpenBSD, FreeBSD, NetBSD, MacOS X, and Windows family tend to respond with MNWNNT.

[99] Few listed exceptions are
Windows 98SE + IE5.5sp1: T3(Resp=N), PU(Resp=N|Y)
Windows NT 4.0 SP 6a + hotfixes : PU(Resp=N|Y)
Windows XP Professional Release candidate 1 or 2: T2(Resp=N), T3(Resp=N), T7(Resp=N)

[100] In rare cases, some of these systems do answer to T2.

## OS scan with Xprobe

Xprobe fingerprinting technique interrogates the target machine's ICMP protocol implementation by sending it craft packets and observing the responses it gets back. Xprobe sends at most four packets. The program of the version examined is built as a decision tree, and does not use a look-up fingerprint file. A graphical representation of this logic tree is given at the end of this section. The version examined during this study was xprobe1-0.0.2. It is this version that is described below. As of November 2004, the most recent version of xprobe is xprobe2-0.2. The underlying techniques to identify the OS are essentially the same, but the tool now relies on fuzzy signature matching and probabilistic guesses. Some additional fields are also examined in response packets.

Xprobe's technique is stealthier than Nmap's. The packets it sends out can appear as being part of normal traffic operations. It is also an alternative to methods that depend solely on the differences between TCP protocol implementations. The TCP stacks of Microsoft based operating systems are very similar and therefore it is often quite difficult to differentiate between them using TCP fingerprinting techniques.

### Description of tests based on the UDP packet

Just like the PU test on Nmap, this test consists on sending a UDP datagram to a closed UDP port (32132 by default) in order to trigger an ICMP *port unreachable* error message in response.

The IP total length of the crafted packet is 98 bytes ( 20B of IP header + 8B of UDP header +70B of data). The DF bit flag is set. The 70 bytes of data carried consist of all zeros.

Note: When a closed UDP port receives a packet, an ICMP Port Unreachable error message is generated. If the port is open, no reply is generated since UDP is a stateless protocol. However, when a filtering device is blocking UDP traffic aimed at a target, the packet will also remained unanswered. Xprobe assumes a closed UDP destination port, and interprets a non-response to this crafted packet as being the result of the presence of a device filtering that port[6].

Xprobe then captures and analyses the response (if any) of the target. More specifically, this first packet allows Xprobe to conduct the following criterion checks:

1. Did we receive a response to this packet?

2. Was the *Don't Fragment* bit set in the target's response?

3. What is the IP *Time To Live* value in the target's response?

4. What Type of service was specified in the target's response? (default? echoed? Other?)

5. What is the amount of the data being echoed with the target's response?

6. Were the following fields of the echoed message altered?

    - IP total length field
    - IP Identification field
    - IP *Flags* and *Offset* field
    - IP checksum field
    - UDP checksum field

Note that since Xprobe follows the branches of a logic tree, the program may stops before completing all checks.

## ICMP Echo Request packet

This test is an ICMP Echo request message sent to probe an ICMP Echo reply message in response.

The IP total length of the crafted packet is 68 bytes ( 20B of IP header + 8B of ICMP header + 40B of data). The 40 bytes of data carried consist of all zeros. The DF bit flag is set. The TOS 8-bit field value is set to "00000110". The ICMP type is 8 but the code is nonzero (which is not standard). This is a trick to differentiate the OSs that automatically zero-out this field from those who echo back the value in their ICMP Echo Reply.

Xprobe then captures and analyses the response (if any) of the target. More specifically, this packet allows Xprobe to conduct the following criterion checks:

1. Did we receive a response to this packet?

2. Was the *Don't Fragment* bit set in the target's response?

3. Was the IP *Identification* field set to zero?

4. What is the IP *Time To Live* value in the target's response?

5. What Type of service was specified in the target's response?

6. What is the ICMP code in the target's response?

## ICMP Timestamp Request

Xprobe sends this packet only to test whether or not it gets a response back. Thus the only criteria is:

1. Did we receive a response to this packet?

## ICMP Information Request

Xprobe sends this packet only to test whether or not it gets a response back. Thus the only criteria is:

1. Did we receive a response to this packet?

## ICMP Address Mask Request

Xprobe sends this packet only to test whether or not it gets a response back. Thus the only criteria is:

1. Did we receive a response to this packet?

## General Remarks on OS differences

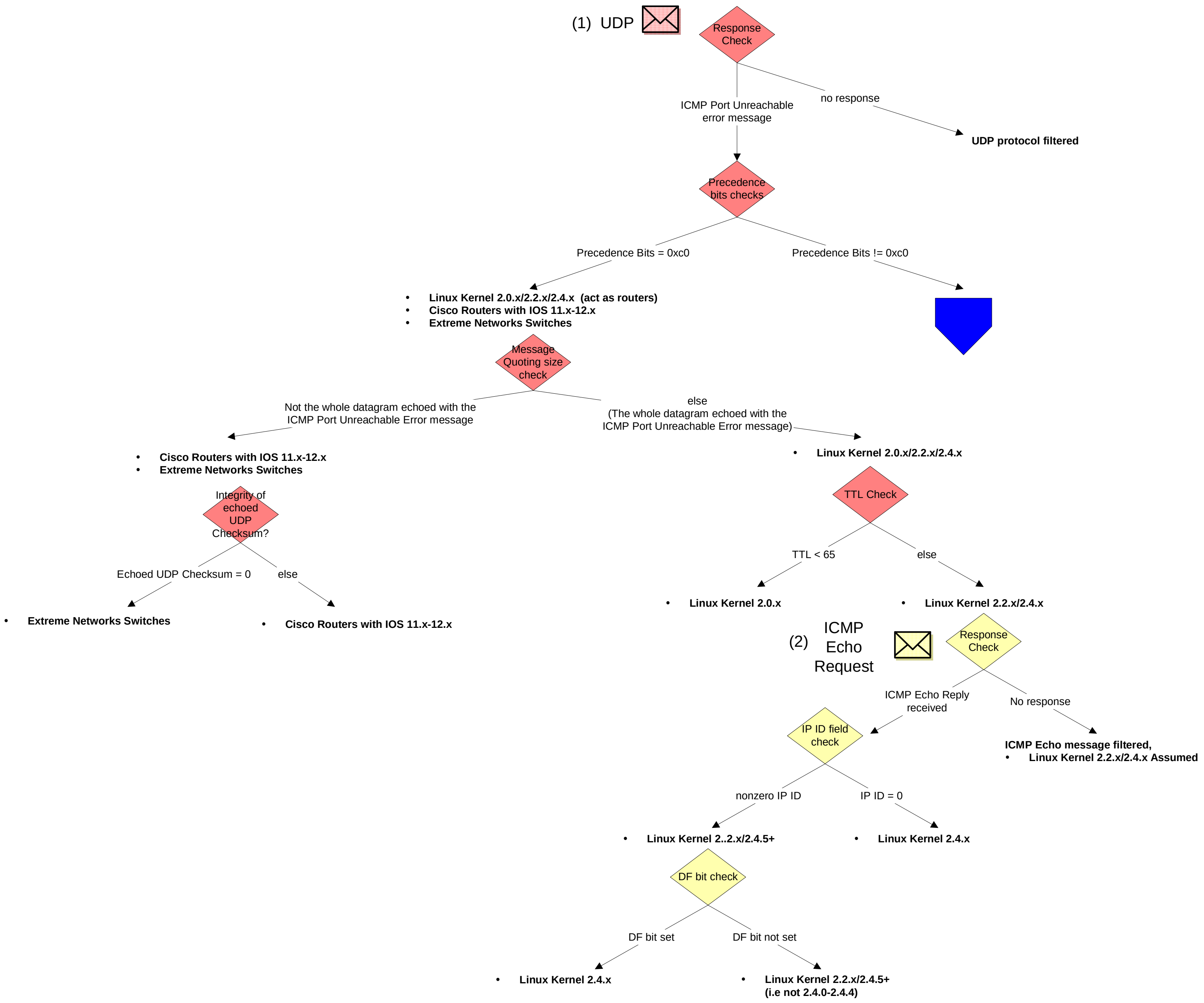
Windows family systems can be detected using one ICMP echo Request. It suffices to set the ICMP code field to a nonzero value in the request packet. In their ICMP Echo Reply, Windows systems overwrite this nonzero value to set the field to zero while other systems just echo it.

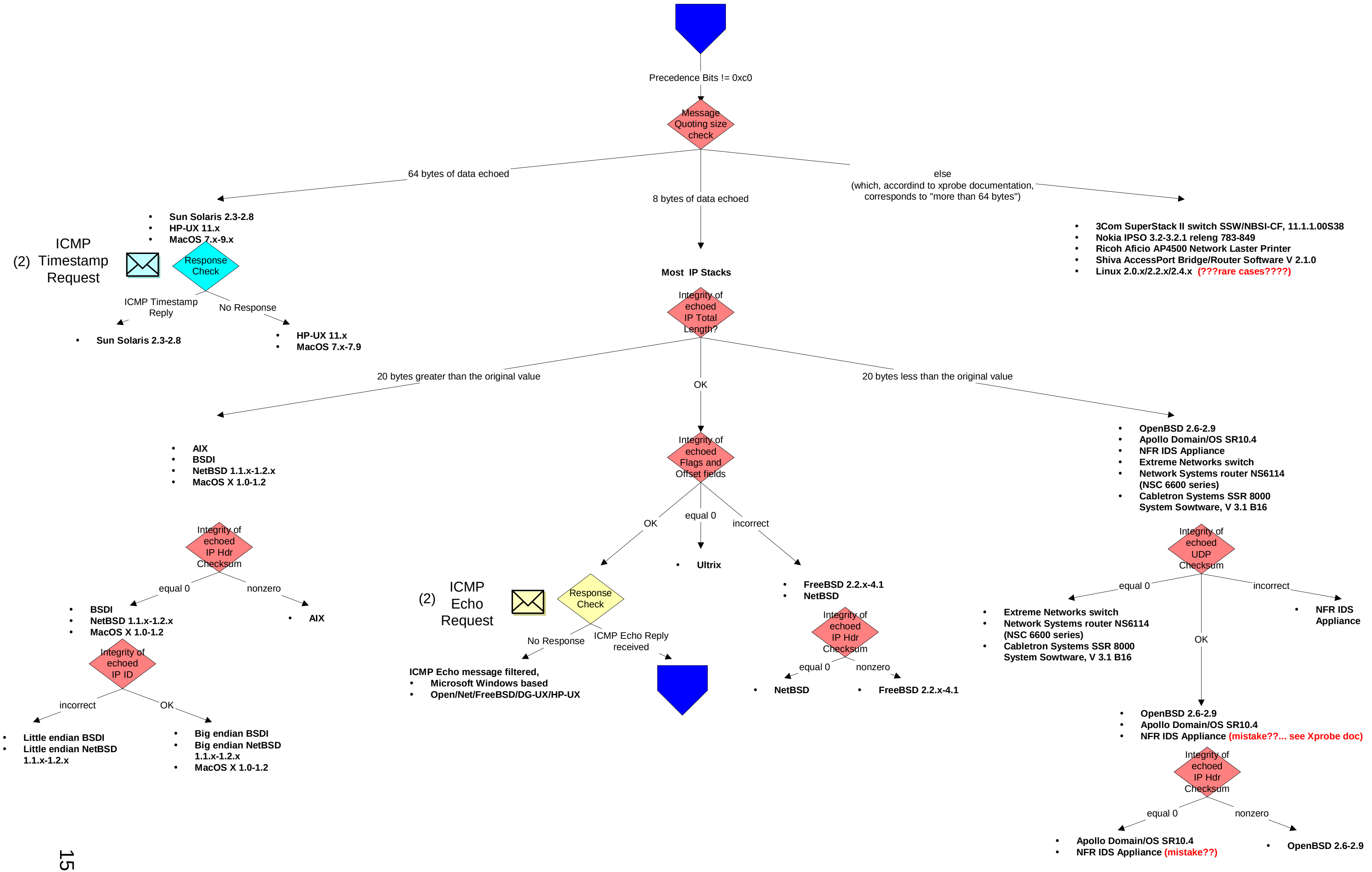
## Graphical representation of Xprobe's logic tree

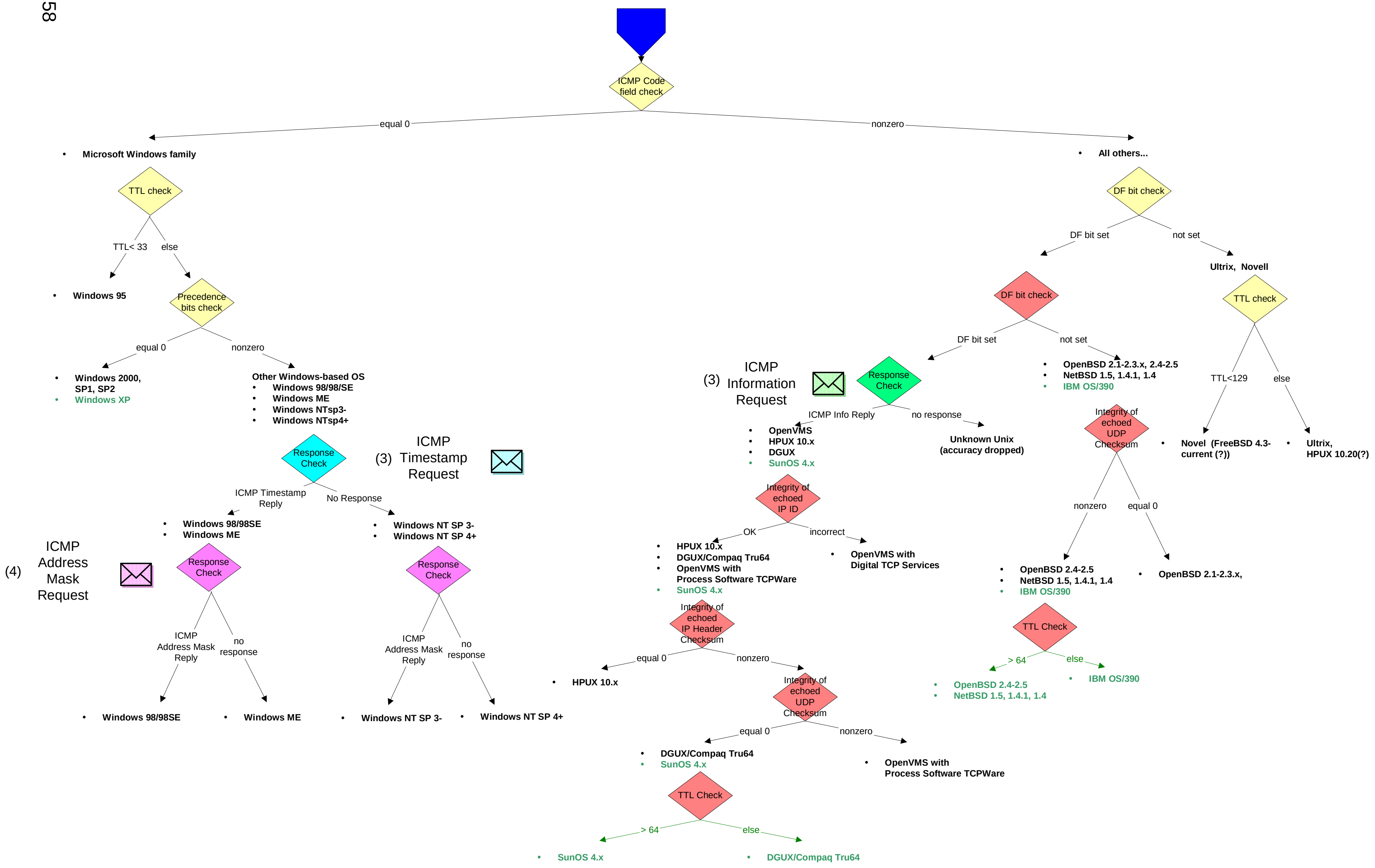
A scheme representing Xprobe's logic tree is shown in the following four pages. This was drawn from examining the program code of xprobe1-0.0.1 and xprobe1-0.0.2.

## Legend

| | |
|---|---|
| ✉ | Xprobe sending a packet |
| ⬟ | Indicates that the branch of the tree continues on next page |
| ◆ | Checks based on response to UDP Packet |
| ◆ | Checks based on response to ICMP Echo Request Packet |
| ◆ | Checks based on response to ICMP Timestamp Request Packet |
| ◆ | Checks based on response to ICMP Information Request Packet |
| ◆ | Checks based on response to ICMP Address Mask Request Packet |
| Green Text | Upgrade from version 0.0.1p1 to 0.0.2 |


(1) UDP
Response Check
ICMP Port Unreachable error message
no response
UDP protocol filtered
Precedence bits checks
Precedence Bits = 0xc0
Precedence Bits != 0xc0
• Linux Kernel 2.0.x/2.2.x/2.4.x (act as routers)
• Cisco Routers with IOS 11.x-12.x
• Extreme Networks Switches
Message Quoting size check
Not the whole datagram echoed with the ICMP Port Unreachable Error message
else (The whole datagram echoed with the ICMP Port Unreachable Error message)
• Cisco Routers with IOS 11.x-12.x
• Extreme Networks Switches
• Linux Kernel 2.0.x/2.2.x/2.4.x
Integrity of echoed UDP Checksum?
Echoed UDP Checksum = 0
else
• Extreme Networks Switches
• Cisco Routers with IOS 11.x-12.x
TTL Check
TTL < 65
else
• Linux Kernel 2.0.x
• Linux Kernel 2.2.x/2.4.x
(2) ICMP Echo Request
Response Check
ICMP Echo Reply received
No response
ICMP Echo message filtered,
• Linux Kernel 2.2.x/2.4.x Assumed
IP ID field check
nonzero IP ID
IP ID = 0
• Linux Kernel 2..2.x/2.4.5+
• Linux Kernel 2.4.x
DF bit check
DF bit set
DF bit not set
• Linux Kernel 2.4.x
• Linux Kernel 2.2.x/2.4.5+ (i.e not 2.4.0-2.4.4)


Precedence Bits != 0xc0
Message Quoting size check
64 bytes of data echoed
8 bytes of data echoed
else
(which, accordind to xprobe documentation, corresponds to "more than 64 bytes")
Sun Solaris 2.3-2.8
HP-UX 11.x
MacOS 7.x-9.x
ICMP
(2) Timestamp
Request
Response Check
ICMP Timestamp Reply
No Response
Sun Solaris 2.3-2.8
HP-UX 11.x
MacOS 7.x-7.9
Most IP Stacks
3Com SuperStack II switch SSW/NBSI-CF, 11.1.1.00S38
Nokia IPSO 3.2-3.2.1 releng 783-849
Ricoh Aficio AP4500 Network Laster Printer
Shiva AccessPort Bridge/Router Software V 2.1.0
Linux 2.0.x/2.2.x/2.4.x (???rare cases????)
Integrity of echoed IP Total Length?
20 bytes greater than the original value
OK
20 bytes less than the original value
AIX
BSDI
NetBSD 1.1.x-1.2.x
MacOS X 1.0-1.2
Integrity of echoed Flags and Offset fields
OpenBSD 2.6-2.9
Apollo Domain/OS SR10.4
NFR IDS Appliance
Extreme Networks switch
Network Systems router NS6114 (NSC 6600 series)
Cabletron Systems SSR 8000 System Sowtware, V 3.1 B16
Integrity of echoed IP Hdr Checksum
equal 0
nonzero
BSDI
NetBSD 1.1.x-1.2.x
MacOS X 1.0-1.2
AIX
Integrity of echoed IP ID
incorrect
OK
Little endian BSDI
Little endian NetBSD 1.1.x-1.2.x
Big endian BSDI
Big endian NetBSD 1.1.x-1.2.x
MacOS X 1.0-1.2
OK
equal 0
incorrect
Ultrix
FreeBSD 2.2.x-4.1
NetBSD
(2) ICMP Echo Request
Response Check
No Response
ICMP Echo Reply received
ICMP Echo message filtered,
Microsoft Windows based
Open/Net/FreeBSD/DG-UX/HP-UX
Integrity of echoed IP Hdr Checksum
equal 0
nonzero
NetBSD
FreeBSD 2.2.x-4.1
Integrity of echoed UDP Checksum
equal 0
incorrect
OK
Extreme Networks switch
Network Systems router NS6114 (NSC 6600 series)
Cabletron Systems SSR 8000 System Sowtware, V 3.1 B16
NFR IDS Appliance
OpenBSD 2.6-2.9
Apollo Domain/OS SR10.4
NFR IDS Appliance (mistake??... see Xprobe doc)
Integrity of echoed IP Hdr Checksum
equal 0
nonzero
Apollo Domain/OS SR10.4
NFR IDS Appliance (mistake??)
OpenBSD 2.6-2.9


ICMP Code field check
equal 0
nonzero
Microsoft Windows family
TTL check
TTL< 33
else
Windows 95
Precedence bits check
equal 0
nonzero
Windows 2000, SP1, SP2
Windows XP
Other Windows-based OS
Windows 98/98/SE
Windows ME
Windows NTsp3-
Windows NTsp4+
Response Check
(3) ICMP Timestamp Request
ICMP Timestamp Reply
No Response
Windows 98/98SE
Windows ME
Windows NT SP 3-
Windows NT SP 4+
(4) ICMP Address Mask Request
Response Check
ICMP Address Mask Reply
no response
Windows 98/98SE
Windows ME
Response Check
ICMP Address Mask Reply
no response
Windows NT SP 3-
Windows NT SP 4+
All others...
DF bit check
DF bit set
not set
Ultrix, Novell
TTL check
TTL<129
else
Novel (FreeBSD 4.3-current (?))
Ultrix, HPUX 10.20(?)
DF bit check
DF bit set
not set
OpenBSD 2.1-2.3.x, 2.4-2.5
NetBSD 1.5, 1.4.1, 1.4
IBM OS/390
Integrity of echoed UDP Checksum
nonzero
equal 0
OpenBSD 2.4-2.5
NetBSD 1.5, 1.4.1, 1.4
IBM OS/390
OpenBSD 2.1-2.3.x,
TTL Check
> 64
else
OpenBSD 2.4-2.5
NetBSD 1.5, 1.4.1, 1.4
IBM OS/390
(3) ICMP Information Request
Response Check
ICMP Info Reply
no response
Unknown Unix (accuracy dropped)
OpenVMS
HPUX 10.x
DGUX
SunOS 4.x
Integrity of echoed IP ID
OK
incorrect
HPUX 10.x
DGUX/Compaq Tru64
OpenVMS with Process Software TCPWare
SunOS 4.x
OpenVMS with Digital TCP Services
Integrity of echoed IP Header Checksum
equal 0
nonzero
HPUX 10.x
Integrity of echoed UDP Checksum
equal 0
nonzero
DGUX/Compaq Tru64
SunOS 4.x
OpenVMS with Process Software TCPWare
TTL Check
> 64
else
SunOS 4.x
DGUX/Compaq Tru64

This page intentionally left blank

# List of symbols/abbreviations/acronyms/initialisms

| | |
|---|---|
| ARP | Address Resolution Protocol |
| CRC | Communications Research Centre |
| DND | Department of National Defence |
| DNS | Domain Name System |
| DRDC | Defence Research and Development Canada |
| ICMP | Internet Control Message Protocol |
| ID | Identifier |
| IP | Internet Protocol |
| IPSec | IP Security Protocol |
| ISN | Initial Sequence Number |
| LAN | Local Area Network |
| NAT | Network Address Translation |
| NIDS | Network Intrusion Detection System |
| OS | Operating System |
| OSes | Operating Systems (plurial) |
| RFC | Request For Comments |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| VPN | Virtual Private Network |
| WLAN | Wireless Local Area Network |

# Glossary

Active/Passive information gathering

In networking, information gathering refers to the process of collecting information about the network and its components. The term active refers to methods that inject traffic (probes) into the network. The term passive indicate that the process is based on methods that silently monitor the network to collect information.

Network Intrusion Detection System (NIDS)

NIDS sensors are deployed in strategic locations within the network infrastructure to monitor network traffic passively in order to detect attacks and intrusions.

OS fingerprinting:

Identification of operating systems by comparing key features of an observed behaviour with known signatures (patterns). This process is similar to identifying an unknown person by taking his or her unique fingerprints and finding a match in a database of known fingerprints.

packet

A small, self-contained parcel of data sent across a computer network. Each packet contains headers of encapsulated protocols and data to be delivered. One of the header identifies the sender and the recipient.

protocol

A design that specifies the details of how computers interact, including the format of messages they exchange and how errors are handled.

TCP/IP stack

TCP/IP stack refers to the implementation of the protocol suite used in the Internet. Although the suite contains many protocols, TCP and IP are two of the most important.

# DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Communications Research Centre Canada
3701 Carling Avenue
Ottawa, Ontario, CANADA, K2H 8S2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

A Multi-Packet Signature Approach to Passive Operating System Detection (U)

4. AUTHORS (Last name, first name, middle initial)

De Montigny-Leboeuf, Annie

5. DATE OF PUBLICATION (month and year of publication of document)

January 2005

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

177

6b. NO. OF REFS (total cited in document)

33

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Joint CRC/DRDC Technical Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence R&D Canada - Ottawa
3701 Carling Avenue
Ottawa, Ontario, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

15bf

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

CRC-TN-2005-001

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

DRDC Ottawa TM 2005-018

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

( x ) Unlimited distribution
(   ) Distribution limited to defence departments and defence contractors; further distribution only as approved
(   ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
(   ) Distribution limited to government departments and agencies; further distribution only as approved
(   ) Distribution limited to defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

13. ABSTRACT ( a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Remote operating system discovery can provide valuable contextual information regarding the computers connected to the network. In particular, operating system discovery can help identify potential vulnerable computers or may help prioritize alarms and responses in times of attack. The Network Security Research Group at the Communication Research Centre (CRC) has developed novel techniques for passive operating system discovery. The methodology developed allows derivation of a signature from a set of packets. The tests are conducted passively on regular traffic. They are non-intrusive and do not rely on access to application or user data. Because they are passive, the techniques do not consume bandwidth and do not disrupt network assets. Over a dozen tests have been developed to analyse headers of packets seen on a network. The tests are conducted on headers of various types of protocols: ARP, IP, ICMP, UDP and TCP. This document describes the tests in detail. They have been implemented in a prototype written in JAVA, which includes a database containing the “fingerprints” of almost 200 versions of operating systems. The prototype was used to collect these signatures from our testbed and was also used on real user traffic for preliminary evaluation of the tests’ performance.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

passive network traffic monitoring, operating system fingerprinting, multi-packet signatures